Tâches de raisonnement en logiques hybrides

Guillaume Hoffmann

Nancy-Université

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Tâches de raisonnement en logiques hybrides

THÈSE

présentée et soutenue publiquement le 13 décembre 2010

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1
(spécialité informatique)

par

Guillaume Hoffmann

Composition du jury

Rapporteurs :	Stéphane DEMRI	Directeur de Recherche CNRS, ENS Cachan, France
	Gert SMOLKA	Professeur, Saarland University, Sarrebruck, Allemagne
Examinateurs :	Patrick BLACKBURN	Directeur de Recherche INRIA, Nancy, France
	Claude GODART	Professeur, Université de Lorraine, Nancy, France
	Andreas HERZIG	Directeur de Recherche CNRS, IRIT, Toulouse, France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria

Résumé : Tâches de raisonnement en logiques hybrides

Les logiques modales sont des logiques permettant la représentation et l'inférence de connaissances. La logique hybride est une extension de la logique modale de base contenant des nominaux, permettant de faire référence à un unique individu ou monde du modèle. Dans cette thèse nous présentons plusieurs algorithmes de tableaux pour logiques hybrides expressives. Nous présentons aussi une implémentation de ces calculs, et nous décrivons les tests de correction et de performance que nous avons effectués, ainsi que les outils les permettant. De plus, nous étudions en détail une famille particulière de logiques liée aux logiques hybrides : les logiques avec opérateurs de comptage. Nous étudions la complexité et la décidabilité de certains de ces langages.

Mots-clés : Déduction automatique, tableaux, logique modale, logique hybride, analyse en complexité, terminaison, benchmarks

Abstract: Reasoning Tasks for Hybrid Logics

Modal logics are logics enabling representing and infering knowledge. Hybrid logic is an extension of the basic modal logic that contains nominals which enable to refer to a single individual or world of the model. In this thesis, we present several tableaux-based algorithms for expressive hybrid logics. We also present an implementation of these calculi and we describe correctness and performance tests we carried out, and the tools that enable these. Moreover, we study a particular family of logics related to hybrid logics: logics with counting operators. We investigate previous results, and study the complexity and decidability of certain of these languages.

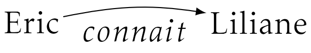
Keywords: Automated deduction, tableaux, modal logic, hybrid logic, complexity analysis, termination, benchmarks

Tâches de raisonnement en logiques hybrides

La déduction automatique est un domaine vaste recouvrant de nombreuses techniques permettant la représentation et l'inférence de connaissances. Ce domaine est enraciné dans celui, bien plus large, de la logique, et il recouvre également le domaine de l'algorithmique et de l'informatique. Depuis des décennies, une entreprise de recherche est menée afin d'améliorer les outils utilisés en déduction automatique, débouchant sur des logiciels pouvant traiter des applications concrètes.

Ces outils sont fondés sur des langages logiques, qui permettent un traitement univoque et un examen exhaustif de leurs propriétés. Cependant, selon la tâche visée, tous les langages ne se valent pas: choisissez le mauvais langage pour votre tâche, et vous n'en ferez jamais quelque chose d'utile. En effet, le choix d'un langage est un compromis, car un grand pouvoir expressif s'accompagne habituellement d'une grande complexité, qui elle n'est pas souhaitable quand on veut implémenter ces outils.

Une option parmi les langages aujourd'hui utilisés pour la représentation et l'inférence sont les *logiques modales*, qui se caractérisent par une sémantique relationnelle et des langages décidables. Ils permettent de raisonner sur des *modèles*, c'est-à-dire des structures représentant un ensemble d'individus, éventuellement non similaires, liés par des relations. Par exemple, la phrase *Éric connait Liliane* est vraie dans le modèle suivant:



La tâche centrale en déduction automatique est de trouver s'il existe, pour une formule donnée, un modèle dans lequel elle soit vraie. C'est ce qu'on appelle la tâche de *satisfiabilité*. Nous venons de voir que la réponse à cette question était "oui" pour *Éric connait Liliane*, mais qu'en est-il de: *Tout le monde est grand et connait quelqu'un* *qui est petit* ? Même si on a une intuition de la réponse dans ce cas précis, comment la *prouver* ?

De plus, comment obtenir la réponse pour des problèmes venant d'applications concrètes, où les formules et modèles en jeu sont bien plus grands et complexes ? Par exemple, dans (Suda et al., 2010), une base de connaissances contenant plus de 10 millions de faits similaires à l'exemple précédent est examinée par un prouveur du premier ordre. Plus communément, le développement de raisonneurs et d'outils d'édition et la standardisation des formats a entrainé la conception des bases de connaissances de taille considérable (Horrocks, 2008).

Si l'on considère la logique propositionnelle, un langage considéré comme simple, son problème de satisfiabilité est pourtant *NP*-complet, ce qui est déjà considéré comme appartenant à une classe de problèmes "difficiles". Or, dans cette thèse, nous allons étudier des algorithmes de déduction automatique pour la satisfiabilité de certains langages modaux, qui sont dans la classe *PSPACE* ou au-delà, et sont donc considérés comme plus difficiles. En conséquence, un soin particulier doit être accordé aux algorithmes de satisfiabilité pour de tels langages.

Cette étude se déroulera de deux manières différentes. D'une part, nous allons présenter des procédures de décision basées sur la méthode des tableaux, dont le but est de donner lieu à des implémentations efficaces. Nous allons mettre ces procédures en action en présentant et évaluant une implémentation, et en comparant ses performances à celles de systèmes existants.

D'autre part, nous allons étudier la décidabilité et la complexité de la tâche de satisfiabilité dans une famille particulière de langages modaux: la logique modale avec opérateurs de comptage. Cette famille peut être vue comme une généralisation d'une autre famille de logiques modales appelée *logiques hybrides*.

Maintenant, regardons plus précisément le paysage dans lequel le travail de cette thèse se situe.

Pourquoi les logiques modales ?

Plantons le décor avec deux logiques très connues: la logique propositionnelle et la logique du premier ordre. Nous allons les considérer comme les deux frontières entre lesquelles se situeront les langages que nous étudierons dans cette thèse.

La **logique propositionnelle** est un langage très simple. Étant donné un ensemble d'atomes propositionnels $\{p_1, p_2, \ldots\}$ (également appelés symboles propositionnels), chacun pouvant être vrai ou faux, on peut écrire des formules telles que $p_1 \vee (\neg p_2 \wedge p_3)$, $\neg$ étant le symbole de la négation et $\vee$ et $\wedge$ les symboles de la disjonction et de la conjonction respectivement. Toute formule propositionnelle est vraie ou fausse selon à la fois sa structure et l'assignement de vérité de ses symboles propositionnels. Par exemple, $p_1 \wedge \neg p_1$ est toujours fausse, $p_1 \vee \neg p_1$ toujours vraie, et $p_1 \wedge p_2$ est vraie sous certaines interprétations et fausses selon d'autres. Si l'on parle en terme de satisfiabilité, $p_1 \wedge \neg p_1$ est insatisfiable tandis que $p_1 \vee \neg p_1$ et $p_1 \wedge p_2$ sont toutes deux satisfiables.

La vérité d'une formule propositionnelle est ainsi fonction de l'assignement de valeurs de vérité des atomes propositionnels qu'elle contient. Vérifier qu'un assignement de valeurs de vérité rend vrai une formule propositionnelle se fait en utilisant les fonctions $(\neg)$, $(\wedge)$ et $(\vee)$, et est donc linéaire en fonction de la taille de la formule, ce qui est peu couteux. En revanche, déterminer la satisfiabilité d'une formule propositionnelle est une tâche *NP*-complète, et les algorithmes couramment utilisés peuvent prendre, dans les cas extrêmes, un temps d'exécution exponentiel en fonction du nombre de symboles propositionnels contenus dans la formule. Ceci parce que, dans le pire des cas, il faut lister et tester chacune des valuations possibles, et il y en a $2^{|\mathrm{Prop}|}$ d'entre elles. Cependant, des optimisations permettent aux implémentations modernes de s'exécuter en un temps acceptable pour la plupart des cas.

Le langage propositionnel est donc très simple, mais cela s'avère être un problème du point de vue du pouvoir expressif dans certains cas. Revenons à notre exemple *Eric connait Liliane*, et essayons de mettre en oeuvre de l'inférence sur des phrases de cette forme en logique propositionnelle. Si nous sommes en présence de n individus Alice, Bob, Charles, ..., il nous faut n^2 symboles propositionnels *Alice-Connait-Bob*, *Alice-Connait-Charles*, ..., *Bob-connait-Alice*, ..., donc le test de satisfiabilité peut nécessiter au pire $c2^{n^2}$ étapes (pour c une constante).

Et encore, ceci est possible car le nombre d'individus est connu à l'avance. Si l'on veut faire de même pour représenter des propriétés dont la vérité dépend du temps ou du lieu, sans avoir à lister les instants ou endroits possibles. D'aucune manière on ne peut représenter des affirmations telles que "aujourd'hui il pleut et demain il ne pleuvra pas, et à partir d'aujourd'hui il pleut" en logique propositionnelle de sorte que la vérité de telles phrases soit calculable de manière fiable en général.

La **logique du premier ordre**, en revanche, se comporte très différemment. On dispose de constantes et de variables interprétées comme des individus parmi un domaine donné, ainsi que de prédicats unaires s'appliquant à des individus isolés, et de relations liant deux ou plusieurs invididus. Un prédicat ou une relation sont vrais ou faux pour un ou plusieurs individus donnés. De plus, on dispose de symboles de fonctions avec lesquels il est possible de construire des références complexes à des individus du domaine. Cependant ces symboles ne sont pas indispensables car peuvent être remplacés par des symboles propositionnels. Il est maintenant possible d'écrire $Grand(Jean)$ pour dire "Jean est grand". Cette formule contient la constante $Jean$ et le prédicat unaire $Grand$.

Il est maintenant également possible de représenter la phrase du paragraphe précédent:

$$\begin{array}{ll} & Pluvieux(Aujourd'hui) \\ \wedge & \neg Pluvieux(Lendemain(Aujourd'hui)) \\ \wedge & \forall x. Futur(Lendemain(x), x) \\ \wedge & \forall x.(Futur(x, Aujourd'hui) \rightarrow Pluvieux(x)) \end{array}$$

Dans cette formule (insatisfiable), $Lendemain$ est un symbole de fonction, $Pluvieux$ est un prédicat unaire et $Future$ un prédicat binaire. x est une variable, qui, avec le quantificateur $\forall$, permet d'exprimer une propriété pour tous les individus du domaine (ici, des jours).

Nous voyons que nous gagnons deux choses avec la logique du premier ordre. D'une part, la *quantification* nous apporte la possibilité de généraliser des affirmations à tous ou certains des individus du domaine. D'autre part, la représentation directe des relations réduit le nombre de symboles utilisés: le symbole $x - Connait - y$ devient $Connait$ applicable à deux termes, ce qui nous donne $n + 1$ symboles au lieu de n^2 symboles.

Malheureusement, ce pouvoir expressif s'accompagne d'un mauvais comportement calculatoire. Pour commencer, cette logique n'a pas la *propriété du modèle fini*, et il est donc impossible d'énumérer les modèles comme on pourrait le faire en logique propositionnelle. Pire, il est possible d'encoder en logique du premier ordre le comportement d'une machine de Turing de sorte que le test de satisfiabilité d'une formule soit équivalent au test de terminaison de la machine de Turing. Ce dernier problème étant indécidable, le premier l'est aussi. Il est donc impossible de concevoir un algorithme, et *a fortiori* une implémentation, qui garantisse la terminaison de la tâche de la satisfiabilité dans ce langage.

Comme les prédicats unaires de la logique du premier ordre se comportent de la même manière que les symboles propositionnels de la logique propositionnelle, on peut établir un lieu de supériorité expressive entre ces deux langages. Faisons-le à l'aide d'une *traduction préservant la satisfiabilité*:

Theorem 1. *Soit* Tr *la fonction de l'ensemble des formules propositionnelles à valeurs dans l'ensemble des formules du premier ordre avec la constante* $Monde$ *et les prédicats unaires* $\{P_1, P_2, \ldots\}$*, définie par:*

$$\begin{array}{rcl} \mathsf{Tr}(p_n) & = & P_n(Monde) \\ \mathsf{Tr}(\varphi \wedge \psi) & = & \mathsf{Tr}(\varphi) \wedge \mathsf{Tr}(\psi) \\ \mathsf{Tr}(\varphi \vee \psi) & = & \mathsf{Tr}(\varphi) \vee \mathsf{Tr}(\psi) \\ \mathsf{Tr}(\neg\varphi) & = & \neg\mathsf{Tr}(\varphi) \end{array}$$

Tr*(φ) est satisfiable si, et seulement si, φ l'est.*

Grâce à cette traduction, nous pouvons ainsi dire que la logique propositionnelle est un *fragment* de la logique du premier ordre, dans le sens où l'on pourrait écrire toute formule de logique propositionnelle avec un fragment de la syntaxe de la logique du premier ordre.

Afin d'obtenir des méthodes de déduction automatique robustes et intéressantes, nous voulons explorer l'espace délimité par ces deux langages. L'existence même de cet espace est suggérée par la traduction que nous venons de voir, et par le saut de complexité entre les deux logiques. En effet, entre les langages *NP*-complets et les langages indécidables, on peut supposer qu'il existe des langages correspondant aux classes intermédiaires: *PSPACE*-complets, *EXPTIME*-complets, ...

Les **logiques modales**, appartiennent à cet espace, et ce sont des langages avec un comportement calculatoire et un pouvoir expressif acceptables. Elles résolvent notamment le problème de pouvoir expressif que nous avions au début de cette section. En effet, nous disposons de symboles propositionnels et de symboles relationnels distincts, et pouvons, avec une formule de la forme $\langle R\rangle\varphi$, exprimer que l'«individu courant», point de départ à partir duquel toute formule est interprétée, est lié par la relation R à un individu tel que φ est vrai. Ainsi nous pouvons exprimer la phrase "aujourd'hui il pleut et ... " par :

$$Pluvieux \wedge (\langle Lendemain\rangle\neg Pluvieux) \wedge \langle Futur\rangle Pluvieux$$

Remarquons que les variables et les quantificateurs n'apparaissent plus: la logique modale n'en a pas besoin. Du point de vue de la sémantique, il y a bel et bien une quantification existentielle exprimée par les connecteurs en diamant tels que $\langle Lendemain\rangle$, mais elle est en quelque sorte restreinte dans son application. Cela a pour conséquence que la famille des langages modaux a un bon comportement calculatoire.

Présentons plus formellement la logique modale de base, à laquelle nous allons nous intéresser plus en détail. Étant donné une signature $\langle \mathsf{PROP}, \mathsf{REL} \rangle$, la grammaire des formules de la logique modale de base est :

$$\varphi \colon = p \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \Diamond_r \varphi$$

avec $p \in \mathsf{PROP}$ et $r \in \mathsf{REL}$. Les connecteurs logiques suivants peuvent être définis en tant que raccourcis : $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\Box_r \varphi \equiv \neg\Diamond_r \neg\varphi$.

La sémantique des formules de logique modale est donnée par des *modèles de Kripke*. Un *modèle* $\mathcal{M}$ est un tuple $\langle W, (R_r)_{r \in \mathsf{REL}}, V \rangle$ avec:

- W un ensemble non vide de mondes, également nommés noeuds ou points
- $R_r \subseteq W \times W$ pour chaque $r \in \mathsf{REL}$, c'est-à-dire, une relation binaire sur W
- $V : \mathsf{PROP} \to 2^W$, une fonction qui associe à chaque symbole propositionnel l'ensemble des mondes dans lesquels il est vrai.

Les modèles de Kripke sont des modèles relationnels, et nous nous restreignons ici à des relations binaires mais il est tout à fait envisageable de faire intervenir des relations n-aires.

Nous pouvons maintenant définir la vérité des formules de logiques modales dans les modèles de Kripke, écrite $\mathcal{M}, w \models \varphi$ pour "φ est vraie dans le modèle $\mathcal{M}$ au monde w":

$$\begin{array}{rll} \mathcal{M}, w \models p & \text{ssi} & w \in V(p), \text{ avec } p \in \mathsf{PROP} \\ \mathcal{M}, w \models \neg\varphi & \text{ssi} & \text{non } \mathcal{M}, w \models \varphi \\ \mathcal{M}, w \models \varphi \wedge \psi & \text{ssi} & \mathcal{M}, w \models \varphi \text{ et } \mathcal{M}, w \models \psi \\ \mathcal{M}, w \models \Diamond_r \varphi & \text{ssi} & \text{il existe } v \in W \text{ tel que } (w, v) \in R_r \text{ et } \mathcal{M}, v \models \varphi \end{array}$$

Une formule φ est *satisfiable* s'il existe un modèle $\mathcal{M}$ et un monde w tel que $\mathcal{M}, w \models \varphi$.

Il faut remarquer que la logique modale ne permet pas d'exprimer correctement des propriétés concernant des individus *uniques*. Par exemple, si l'on écrit $Eric \wedge \langle connait \rangle Liliane$, on s'attend à ce que $Eric$ et $Liliane$ soient des propriétés vraies pour un seul individu unique du domaine. Ainsi, on aimerait que $Eric \wedge \langle connait \rangle (Liliane \wedge Jeune) \wedge Eric \wedge \langle connait \rangle (Liliane \wedge \neg Jeune)$ soit non satisfiable, plutôt que de permettre qu'Éric connaisse deux Liliane, l'une jeune et l'autre un peu moins. Or, la sémantique d'un symbole propositionnel n'oblige pas celui-ci à être vrai dans un seul monde.

La **logique hybride** résout ce problème. Par définition, la logique hybride est la logique modale enrichie de symboles propositionnels spéciaux appelés **nominaux**. Un nominal a pour particularité d'être vrai en un seul point d'un modèle. C'est donc un pointeur vers un endroit unique du modèle, ou, selon les points de vue, c'est le nom d'un individu précis du modèle.

La grammaire des formules de la logique hybride de base est très similaire à celle de la logique modale de base. Nous nous donnons un ensemble de nominaux NOM inclus dans PROP. La grammaire du langage $\mathcal{H}(:)$ est donnée par:

$$\varphi := p \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \Diamond_r \varphi \mid a : \varphi$$

avec $p \in \mathsf{PROP}$, $r \in \mathsf{REL}$ et $a \in \mathsf{NOM}$. Une formule $a:\varphi$ signifie intuitivement «φ est vrai dans le monde désigné par le nominal a». Sémantiquement, la seule différence, dans la définition d'un modèle, avec la logique modale de base est que tout nominal est vrai dans seulement un monde. Cela se traduit formellement par le fait que $V(a)$ est un singleton si $a \in \mathsf{NOM}$. La sémantique de «:», nommé l'*opérateur de satisfaction*, est donnée par:

$$\mathcal{M}, w \models a : \varphi \quad \text{ssi} \quad \mathcal{M}, v \models \varphi \text{ avec } V(a) = \{v\}$$

La logique modale de base et la logique hybride de base ont un problème de satisfiabilité décidable, dont la complexité est *PSPACE*-complète. L'ajout de nominaux et de l'opérateur «:» à la logique modale ne change ainsi pas la classe de complexité.

Il faut tout de même noter, et cela prend son importance dans la conception d'algorithmes pour la satisfiabilité hybride, que le comportement du langage se complique. Considérons cette propriété vraie pour la logique modale:

Theorem 2 (Propriété du modèle en arbre). *Soit φ une formule de logique modale de base contenant une seule modalité. Si φ est satisfiable, alors il existe un modèle $M = \langle W, R, V \rangle$ et $w \in W$ tels que $M, w \models \varphi$ et le graphe $\langle W, R \rangle$ est un arbre de racine w.*

Alors que la logique modale jouit de la propriété du modèle en arbre, la logique hybride peut décrire des modèles non-arbres. Si $a \in \mathsf{NOM}$ et $p \in \mathsf{PROP}$, alors la formule $\Diamond(p \wedge \Diamond a) \wedge \Diamond(\neg p \wedge \Diamond a)$ décrit un modèle en forme de diamant, et la formule $a \wedge \Diamond a$ une boucle.

La perte de la propriété du modèle en arbre signifie que tester la satisfiabilité d'une formule en tentant d'exhiber un modèle dans laquelle elle serait vraie devient une tâche plus compliquée.

La logique hybride de base peut, à l'instar de la logique modale de base, être enrichie de nouveaux opérateurs pour augmenter son pouvoir expressif (mais également souvent sa complexité). Nous donnons tout de suite la sémantique de nouveaux opérateurs que nous retrouvons tout au long de cette thèse:

$$\begin{array}{rcl} \mathcal{M}, w \models \mathsf{E}\varphi & \text{ssi} & \text{il existe } v \in W \text{ tel que } \mathcal{M}, v \models \varphi \\ \mathcal{M}, w \models \mathsf{D}\varphi & \text{ssi} & \text{il existe } v \neq w \text{ et } \mathcal{M}, v \models \varphi \\ \mathcal{M}, w \models \Diamond_r^- \varphi & \text{ssi} & \text{il existe } v \in W, (v, w) \in R_r \text{ tel que } \mathcal{M}, v \models \varphi \end{array}$$

Ces trois nouveaux opérateurs sont respectivement nommés la *modalité globale existentielle*, la *modalité de différence* et la *modalité inverse*. Chacun donne lieu à un opérateur dual, qui sont respectivement A, B et $\Box_r^-$.

Le langage modal le plus complet que nous connaissons maintenant, à savoir la logique hybride équipée de E, D, et $\Diamond^-$, reste décidable, et sa satisfiabilité est *EXPTIME*-complète.

Par souci de concision, nous nommons les différents langages rencontrés dans cette thèse avec la convention suivante. Si le langage modal en question contient des nominaux nous l'appelons $\mathcal{H}$, sinon $\mathcal{M}$, et nous attachons au nom les opérateurs supplémentaires. Par exemple, nous rencontrerons $\mathcal{H}(:, \mathsf{E})$ et $\mathcal{M}(\Diamond^-)$.

Nous allons maintenant détailler les contributions de cette thèse en trois parties. Tout d'abord nous présentons les méthodes de tableaux pour logiques hybrides que nous avons améliorées. Ensuite, nous présentons notre étude d'une famille inhabituelle de logiques modales, les logiques modales avec opérateurs de comptage, avec notamment l'accent sur l'aspect calculatoire et le lien avec les logiques hybrides. Enfin, nous présentons notre travail portant sur l'implémentation de nos méthodes de tableaux pour logiques hybrides, ainsi qu'une étude de la méthodologie adoptée pour évaluer et tester cette implémentation.

Tableaux pour logiques hybrides

Une première contribution de cette thèse est l'amélioration de la méthode des tableaux préfixée pour logique hybride proposée par Bolander and Blackburn (2007a).

Les méthodes des tableaux sont une famille d'algorithmes servant à tester la satisfiabilité des formules d'une logique donnée. Elles fonctionnent en essayant de construire un modèle pour la formule, en se guidant sur la structure de celle-ci. La structure d'une formule logique est donnée par son connecteur principal et les sous-formules reliées par ce connecteur. Ainsi, pour accomplir cette construction, une méthode des tableaux doit comporter un ensemble de règles, chacune dédiée à la décomposition d'un type de formules déterminé par son connecteur principal. La formule est réduite en atomes, tandis que le modèle est construit en fonction des règles appliquées.

Dans le cas de la logique modale, une manière courante de définir une méthode des tableaux consiste à utiliser des préfixes pour noter les mondes dans lesquels les sous-formules de la formule initiale sont vraies. Par exemple, si la formule initiale est $\Diamond\varphi$, on la place au préfixe σ_0 en notant $\sigma_0\Diamond\varphi$. Puis on procède à l'application de la règle pour le connecteur $\Diamond$ en introduisant le préfixe σ_1 et en notant $\sigma_1\varphi$ ainsi que $\sigma_0\Diamond\sigma_1$ pour dire que les mondes désignés par σ_0 et σ_1 sont reliés par la relation d'accessibilité. La formule préfixée $\sigma_1\varphi$ doit ensuite être traitée par la règle correspondant au connecteur principal de φ, ou aucune si φ est un littéral, c'est-à-dire une formule de la forme p ou $\neg p$ avec $p \in \mathsf{PROP}$.

Si une contradiction est découverte, ici de la forme σp et $\sigma\neg p$ pour un préfixe σ et un symbole propositionnel p, alors on conclut que la formule initiale est insatisfiable. Au contraire, si on ne peut plus appliquer de règle sans se répéter et qu'il n'y a pas de contradiction, la formule est satisfiable. Si de plus, l'algorithme est garanti terminant pour toute entrée, alors on dit que c'est une *procédure de décision*.

La méthode de Bolander et Blackburn est une procédure de décision pour le langage $\mathcal{H}(:, \mathsf{E}, \Diamond^-)$ de type méthode des tableaux. Elle fut la première méthode des tableaux pour logique hybride garantissant la terminaison pour le langage $\mathcal{H}(:)$ sans recourir à une condition globale de blocage d'application des règles (*loop-check*).

Cependant, ce calcul était perfectible, notamment dans sa manière de gérer l'égalité rendue possible par les nominaux. En effet, dès lors qu'un nominal apparait à droite de plusieurs préfixes, ceux-ci doivent désigner le même monde dans le modèle en cours de construction. Pour cette raison, les formules préfixées par ceux-ci doivent être mises en commun. Or, dans la méthode de Bolander et Blackburn, ceci se traduisait par la copie de toutes les formules de chaque préfixe à tous les autres préfixes d'une même classe d'équivalence. Pourtant, il est connu que le problème de la représentation de classes d'équivalences, aussi appelé «union-find», peut être traité efficacement avec une structure de données de type forêt d'arbres dont la racine est le représentant de sa classe d'équivalence (Cormen et al., 2001). Transposée dans le contexte de la méthode des tableaux pour logique hybride, cette idée consiste à ne copier les formules que vers un seul préfixe. Nous avons donc pu simplifier le jeu de règles de la méthode des tableaux tout en conservant les bonnes propriétés du calcul.

Autre modification, notre calcul initialise autant de préfixes que de nominaux apparaissant dans la formule, faisant que la règle qui traite le connecteur : n'a plus besoin de créer de nouveaux préfixes.

Nous avons également étendu ce calcul pour qu'il gère la modalité de différence D, en suivant la méthode proposée par Kaminski and Smolka (2009b). En effet, leur méthode fut proposée après celle de Bolander et Blackburn, et fut la première à gérer correctement la modalité de différence en logique modale.

Nous présentons notre calcul en trois versions incrémentales. La première version est une procédure de décision pour la logique hybride $\mathcal{H}(\mathsf{D})$ sur modèles dont la relation d'accessibilité est une relation d'équivalence. La seconde est une procédure de décision pour la logique $\mathcal{H}(\mathsf{D})$ sur modèles arbitraires, mais pouvant optionnellement posséder des relations réflexives ou transitives. Cette version utilise un *loop-check* basé sur le blocage des préfixes par inclusion de l'ensemble des formules qu'ils rendent vraies. Enfin, la dernière version est une procédure de décision pour la logique $\mathcal{H}(\mathsf{D}, \Diamond^{-})$ sur modèles arbitraires, avec des relations pouvant être réflexives, transitives ou symétriques. Cette version, à l'instar du calcul de Bolander et Blackburn et de celui de Kaminski et Smolka pour la logique modale avec modalité inverse, nécessite un *loop-check* différent basé sur l'égalité de l'ensemble des formules de préfixes appartenant à une même chaine de parenté.

Décrivons ces trois logiques:

1. le langage $\mathcal{H}(\mathsf{D})$ interprété sur les modèles ayant une relation d'accessibilité qui est une relation d'équivalence, dont les formules sont données par la grammaire:

$$\varphi \ ::= \ p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Diamond\varphi \mid \Box\varphi \mid a : \varphi \mid \mathsf{E}\varphi \mid \mathsf{A}\varphi \mid \mathsf{D}\varphi \mid \mathsf{B}\varphi$$

   avec $p \in \mathsf{PROP}$ et $a \in \mathsf{NOM}$.

2. le langage $\mathcal{H}(\mathsf{D})$ interprété sur des modèles arbitraires, avec éventuellement des relations réflexives et transitives, dont les formules sont données par la gram-

maire:

$$\varphi ::= p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Diamond_i \varphi \mid \Box_i \varphi \mid a : \varphi \mid \mathsf{E}\varphi \mid \mathsf{A}\varphi \mid \mathsf{D}\varphi \mid \mathsf{B}\varphi$$

avec $p \in \mathsf{PROP}$, $a \in \mathsf{NOM}$, $i \in \mathsf{REL}$.

3. le langage $\mathcal{H}(\mathsf{D}, \Diamond^-)$ interprété sur des modèles arbitraires, avec éventuellement des relations réflexives, transitives et symétriques, dont les formules sont données par la grammaire:

$$\begin{aligned}\varphi ::= {} & p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Diamond_i \varphi \mid \Box_i \varphi \mid \Diamond_i^- \varphi \mid \Box_i^- \varphi \\ & \mid a : \varphi \mid \mathsf{E}\varphi \mid \mathsf{A}\varphi \mid \mathsf{D}\varphi \mid \mathsf{B}\varphi\end{aligned}$$

avec $p \in \mathsf{PROP}$, $a \in \mathsf{NOM}$ et $i \in \mathsf{REL}$.

Cette présentation incrémentale permet de voir quelles sont les précautions à observer pour chaque logique afin de préserver les propriétés de terminaison, complétion et correction.

Enfin, nous mentionnons des possibilités d'extension de ces calculs mais qui ne garantissent plus la terminaison. Les trois extensions proposées sont la gestion des hiérarchies de rôles, c'est-à-dire spécifier qu'une relation du modèle est incluse dans une autre, la gestion des relations fonctionnelles et injectives et enfin la gestion du lieur $\downarrow$. Il est connu que la logique hybride $\mathcal{H}(\downarrow)$ est indécidable. Ces extensions sont complètes et correctes mais fonctionnent en supprimant toute *loop-check*.

Langages modaux avec opérateurs de comptage

La seconde contribution de cette thèse est l'étude d'une famille de langages relativement inhabituels sous l'angle de la déduction automatique.

La logique de cette famille qui nous intéresse est la logique modale équipée d'opérateurs de comptage $\varphi \geq n$ et $\varphi \leq n$, permettant de stipuler des conditions de comptage sur l'ensemble du modèle. Leur sémantique est donnée comme suit:

$$\begin{aligned}\mathcal{M}, w \models \varphi \geq n \quad & \iff \quad |\{v \mid \mathcal{M}, v \models \varphi\}| \geq n \\ \mathcal{M}, w \models \varphi \leq n \quad & \iff \quad |\{v \mid \mathcal{M}, v \models \varphi\}| \leq n\end{aligned}$$

Pour avoir une meilleure perspective sur ce langage, nous avons préféré commencer par étudier les langages existants faisant intervenir des opérateurs de comptage. Il s'avère que ces opérateurs ont été mainte fois ajoutés à diverses logiques.

Nous considérons ainsi la logique du premier ordre avec quantificateurs généralisés, avec notamment les fragments avec une et deux variables et opérateurs de comptage, qui sont décidables.

Puis nous considérons la logique modale avec comptage des successeurs ou *graded modal logic*. Cette logique comprend les opérateurs suivants:

$$\begin{array}{lcl} \mathcal{M}, w \models \langle r \rangle_{\geq n} \varphi & \iff & |\{w' \mid R(w,w') \text{ and } \mathcal{M}, w' \models \varphi\}| \geq n \\ \mathcal{M}, w \models \langle r \rangle_{\leq n} \varphi & \iff & |\{w' \mid R(w,w') \text{ and } \mathcal{M}, w' \models \varphi\}| \leq n \end{array}$$

Cette logique permet de compter parmi les successeurs du monde courant. Nous voyons que faire ceci dans un modèle dont la relation d'accessibilité est une relation totale permet de compter globalement. Cette idée a déjà été suggérée par Fine (1972) et van der Hoek and de Rijke (1993, 1995). Cependant, se restreindre à une telle logique ne permet pas d'exprimer de relation entre les individus comptés: les relations d'accessibilité disparaissent.

Enfin, nous considérons les logiques de description: cette fois, ces opérateurs ont été introduits sous la forme d'axiomes spécifiant des contraintes de comptage globales sur des concepts.

De cet état de l'art, nous concluons que l'étude de la logique modale de base avec pour unique extension le comptage global n'avait pas encore été explorée. C'est dans ce but que nous introduisons le langage $\mathcal{MLC}$. Nous étudions sa proximité avec $\mathcal{H}(\mathsf{E})$, la logique hybride équipée de la modalité universelle, et proposons une traduction de $\mathcal{MLC}$ vers celle-ci. Nous étudions également le pouvoir expressif de $\mathcal{MLC}$ en introduisant une bisimulation $\leftrightarrows$ et démontrons les propriétés suivantes:

Theorem 3. *Si $\mathcal{M}, w \leftrightarrows \mathcal{M}', w'$ alors $\mathcal{M}, w$ et $\mathcal{M}', w'$ satisfont les mêmes formules de $\mathcal{MLC}$.*

Theorem 4. *Soit $\mathcal{M} = \langle W, R, V \rangle$ et $\mathcal{M}' = \langle W', R', V' \rangle$ deux modèles finis et $(w, w') \in W \times W'$, $\mathcal{M}, w \leftrightarrows \mathcal{M}', w'$ si, et seulement si, $\mathcal{M}, w \equiv_{\mathcal{MLC}} \mathcal{M}', w'$.*

Équipés de cette bisimulation aux bonnes propriétés, nous montrons également que $\mathcal{MLC}$ et la *graded modal logic* ont un pouvoir expressif incomparable.

Pour finir sur les opérateurs de comptage, nous introduisons une nouvelle tâche d'inférence pouvant être effectuée dans tout langage avec comptage. Cette tâche consiste à déterminer, s'il existe, le nombre n tel qu'un ensemble de formules donné implique logiquement qu'une formule donnée est vraie en n mondes. Nous donnons un algorithme permettant d'effectuer cette tâche de comptage dans tous les langages avec comptage, à condition que l'on dispose, pour le langage en question, d'une procédure de décision de satisfiabilité avec construction de modèle.

Cette nouvelle tâche d'inférence est remarquable car, contrairement à bien d'autres tâches connues comme la validité ou la récupération d'instances (*instance retrieval*), elle ne peut se réduire à la simple tâche de satisfiabilité.

De cette excursion dans un territoire moins connus, nous pouvons en tirer une conclusion : les logiques modales avec comptage sont une généralisation des logiques hybrides.

Implémentations

Le troisième et dernier apport de cette thèse est l'aspect implémentatoire. Comme nous l'avons dit en introduction, la finalité de la déduction automatique est de fournir des outils permettant la représentation et l'inférence sur l'information. Ayant présenté une nouvelle méthode des tableaux pour logique hybride, nous l'avons implémentée dans le logiciel HTab (Hoffmann and Areces, 2009).

HTab réalise la tâche de test de satisfiabilité sur des formules du langage $\mathcal{H}(\mathsf{D}, \Diamond^{-})$ avec possibilité de spécifier des relations comme étant réflexives, transitives ou symétriques. Dans le cas où la formule donnée est satisfiable, HTab peut sortir un modèle dans laquelle elle est vraie. Le prouveur peut également tester si une formule est valide, et peut effectuer la tâche de récupération d'instance. Cette dernière tâche consiste, étant donnée une théorie sous la forme d'un ensemble de formules Γ, et une formule φ, à lister les nominaux n de $\Gamma \cup \{\varphi\}$ tels que $\Gamma \models n : \varphi$. HTab garantit que ces tâches terminent dans cette logique.

Le prouveur contient également des fonctionnalités expérimentales, pour lesquelles la terminaison n'est plus garantie: le lieur $\downarrow$, l'inclusion de roles, les modalités fonctionnelles et injectives. De plus, HTab supporte la modalité fermeture transitive de manière non prouvée (i.e., la correction et la complétion ne sont pas non plus garanties).

Comme toute implémentation d'algorithme de tableaux voulant être performante, HTab inclut des optimisations. Parmi celles-ci on compte la disjonction sémantique, le *backjumping*, la propagation booléenne gloutonne (*eager unit propagation*), et le *lazy branching* (une optimisation relativement récente introduite dans (Götzmann et al., 2010)).

HTab a été le premier prouveur pour logique hybride basé sur la méthode des tableaux dont l'algorithme garantit la terminaison. Avant lui, en 2002, fut écrit HyLoTab, une implémentation décrite par son auteur comme étant un prototype (Eijck, 2002) et ne garantissant pas la terminaison. Peu de temps avant fut commencé le développement de HyLoRes (Areces and Heguiabehere, 2001), un prouveur basé sur la méthode de la résolution pour logique hybride (Areces et al., 2001b; Gorín, 2009).

Depuis HTab, nous avons vu apparaitre Spartacus (Götzmann et al., 2010) basé sur la méthode des tableaux de Kaminski and Smolka (2009b). Spartacus met en oeuvre le *pattern-based blocking*, une technique de blocage qui subsume le blocage par inclusion utilisé dans HTab, et décrite par ses auteurs comme permettant une implémentation plus performante. Ce prouveur met également en oeuvre pour la première fois l'optimisation du *lazy branching*, qui permet d'éviter de brancher dans le tableau autant que nécessaire.

Bien que nous comparâmes HTab à HyLoTab et HyLoRes en 2007 pour évaluer sa performance (Hoffmann and Areces, 2009), nous nous référons désormais à Spartacus (Götzmann et al., 2010) qui est bien plus compétitif et souvent plus rapide que HTab.

Une fois faite la description du prouveur HTab, nous abordons le sujet de l'évaluation. Nous commençons par parler du problème de la couverture des tests de prouveurs: comment savoir que l'on teste un prouveur sur un espace suffisant parmi l'espace de toutes les entrées possibles ? Dans le cas de la logique propositionnelle, on sait que l'on peut passer d'une zone de formules surtout insatisfiables à une zone de formules surtout satisfiables en jouant sur le rapport nombre de clauses/nombres de variables, et ainsi obtenir une couverture satisfaisante.

Dans le cas de la logique modale et hybride, l'approche adoptée est similaire. Nous avons employé un système nommé GridTest (Areces et al., 2009). Nous définissons un test (*benchmark*) comme étant un ensemble de paramètres stipulant le nombre de symboles propositionnels, de nominaux et de modalités pouvant apparaitre dans les formules, mais également la profondeur modale des formules, et enfin une fourchette de taille de formules en nombre de clauses. Nous utilisons le générateur aléatoire de formules hGen (Areces and Heguiabehere, 2003) pour générer une succession de jeux de formules dont le nombre de clauses appartient à cette fourchette. Par exemple, nous pouvons générer des formules avec 10, 20, 30, . . . , 100 clauses. Nous exécutons ensuite un ou plusieurs prouveurs sur chacune des formules de ces 10 jeux. Pour chaque jeu enfin, nous calculons la durée médiane de calcul de chaque prouveur, et affichons ces informations sur un même graphe (voir Figure 0.1).

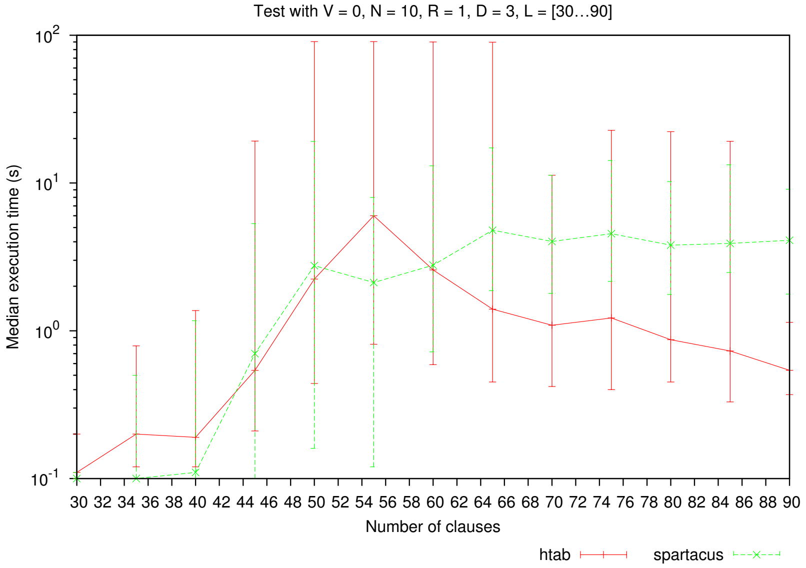


Figure 0.1: Graphe de comparaison du temps médian de réponse des prouveurs

Afin de tester suffisamment le comportement des prouveurs, nous nous assurons que le benchmark fournit des jeux de formules étant d'abord surtout satisfiables puis, avec l'augmentation du nombre de clauses, devenant surtout insatisfiables. Nous pouvons vérifier ceci sur un graphe indiquant cette répartition (voir Figure 0.2).

Le système de benchmark GridTest peut s'exécuter localement sur une seule machine, mais peut aussi se lancer sur une grille d'ordinateurs. Le but est alors d'exécuter les tests en parallèle afin de gagner du temps. En effet, un benchmark peut parfois durer des heures voire des jours, ce qui est pénible quand on recherche les paramètres permettant d'obtenir un benchmark satisfaisant, ou quand on veut mesurer l'effet de diverses combinaisons d'optimisations.

Répartir les calculs sur n machines permet de diviser le temps de benchmark par n. En effet, chaque jeu de formules de même taille est divisé en n sous-paquets envoyés chacun sur une machine. Ceci arrive pour chaque jeu du benchmark. Une fois qu'une machine a terminé les calculs, elle renvoie les résultats à la machine qui supervise la répartition. Quand tous les résultats sont obtenus, ils sont rassemblés dans un seul et même rapport identique à celui généré quand le test est lancé sur une seule machine.

Une autre utilité de GridTest est de découvrir des bogues. Dans le rapport généré à chaque benchmark figure une partie «Incohérences» qui liste les réponses discordantes entre prouveurs. Cela nous a permis de nombreuses fois de découvrir des bogues dans HTab et parfois dans d'autres prouveurs. De ce point de vue, l'usage

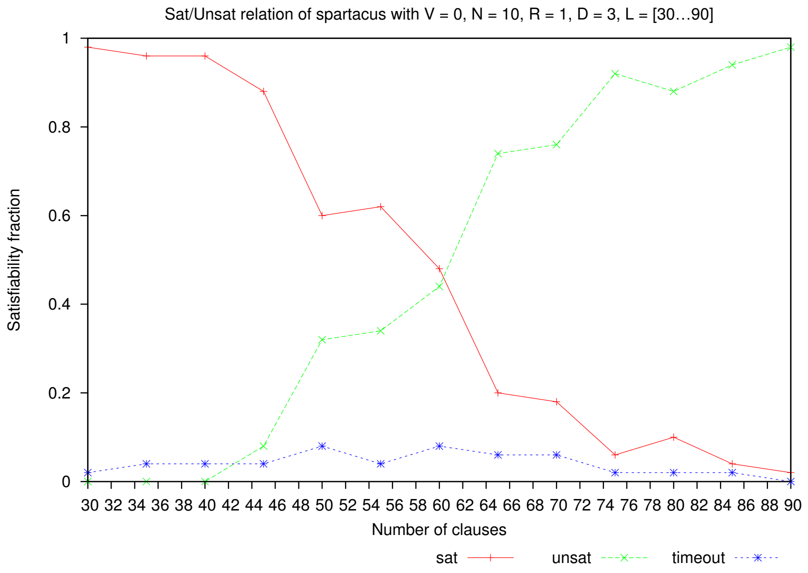


Figure 0.2: Graphe de répartition des proportions de formules satisfiables, insatisfiables, et inconnues (pour cause de limite de temps dépassée) pour un prouveur donné

simultané de GridTest et d'un générateur aléatoire de formules est une forme de débogage nommée *grammar-based blackbox fuzzing* (Brummayer and Biere, 2009).

Plan de la thèse

Pour finir, voyons en détail le contenu de chacun des chapitres à venir:

Chapitre 1 : Contenu de cette thèse Ce chapitre est une version courte de ce résumé en français.

Chapitre 2 : Complexité et logique Ce chapitre donne les bases de la complexité définie à l'aide de machines de Turing. En particulier, nous voyons les définitions habituelles des classes de complexité basées sur des ressources en temps et en espace. Puis, il se tourne vers la définition et un aperçu des propriétés de diverses logiques utiles pour nos investigations: logique modale, logique hybride, logiques de description et logique du premier ordre. Le but est de préparer le terrain pour les résultats qui sont présentés plus loin dans la thèse.

Chapitre 3 : Introduction aux méthodes des tableaux Ce chapitre donne les bases des méthodes des tableaux. Il commence par un survol de cette méthode pour les logiques propositionnelle et modale. Dans le deuxième cas, la méthode avec préfixes est montrée. Puis, on rappelle l'histoire de cette méthode ainsi que les tendances récentes dans cette famille d'algorithmes.

Chapitre 4 : Tableaux préfixés pour logiques hybrides Ce chapitre présente des procédures de décision de type méthodes des tableaux pour plusieurs langages hybrides. Notamment, trois logiques sont vues de façon incrémentale: la logique hybride sur modèles dont le cadre est une relation d'équivalence, la logique hybride sur modèles arbitraires, et enfin la logique hybride sur modèles arbitraires avec modalité inverse. Dans ces trois cas, le calcul proposé est prouvé comme étant adéquat, c'est-à-dire que la terminaison, la complétion et la correction est prouvée. De plus, des modifications de ces calculs sont proposées à la fin de ce chapitre, afin de gérer des extensions de la logique hybride. Ces extensions sont l'inclusion de rôle, le lieur $\downarrow$ et les modalités injectives et fonctionnelles. Dans ces extensions, la terminaison du calcul n'est plus garantie.

Chapitre 5 : Logiques avec comptage Dans ce chapitre, nous abordons la question de l'ajout d'opérateurs de comptage aux logiques du premier ordre, modale et de description. Ces opérateurs permettent d'exprimer des phrases du type «au moins/au plus n objets de type a». Dans chacun des langages considérés, la question a déjà été abordée, et nous faisons un état de l'art détaillé. Nous intoduisons également une logique modale équipée de tels opérateurs, et étudions sa complexité et son pouvoir expressif. Nous mettons en évidence son lien avec la logique hybride et d'autres logiques modales. Pour finir, nous introduisons une nouvelle tâche d'inférence: la tâche consistant à déterminer, s'il existe, le nombre n tel qu'un ensemble de formules donné implique logiquement qu'une formule donnée est vrai en n mondes. Nous donnons un algorithme permettant d'effectuer cette tâche de comptage dans toutes les logiques avec comptage, à condition que l'on dispose, pour la logique en question, d'une procédure de décision de satisfiabilité avec construction de modèle.

Chapitre 6 : Description du logiciel HTab Dans ce chapitre, nous présentons HTab, une implémentation des algorithmes de tableaux présentés dans le Chapitre 4. Nous montrons comment obtenir et utiliser ce prouveur, et discutons des optimisations incorporées. Nous évoquons également les différences entre le calcul implémenté et sa définition. Nous évoquons enfin les autres systèmes en logiques hybrides et logiques de description ayant des fonctionnalités similaires.

Chapitre 7 : Benchmarks Ce dernier chapitre aborde la question qui suit toute implémentation: l'évaluation. Celle-ci est présentée sous l'angle des benchmarks. Nous évoquons les problèmes liés à l'évaluation des prouveurs, et nous présentons le système que nous avons utilisé pour HTab, nommé GridTest. Nous présentons le fonctionnement de ce système, avec notamment la possibilité d'exécuter des tests en parallèle sur les machines d'une grille. Après avoir montré deux exemples de tests, nous terminons par tracer quelques pistes pour l'amélioration de ce système.

Contents

Chapter 1

Contents of this thesis

Automated deduction is a vast domain covering numerous techniques enabling the representation and inference on knowledge. This domain is rooted in logic, and it also adds to the domain of algorithmic and computer science. Since decades, an endeavour of investigation has been led in order to improve tools used for automated deduction, culminating in software able to treat concrete applications.

These tools are based on logical languages, that enable an univocal treatment and an thorough examination of their properties. However, depending on the task aimed at, all languages are not the same: choose the wrong language and you will never be able to make something useful out of it. Indeed, the choice of a language is a compromise, since a great expressive power usually comes with a great complexity, which is never something one wants for creating tools.

As of now, one option for the representation and inference is the family of *modal logics*, that are characterised by their relational semantics and decidable languages. They enable to reason on *models*, that are structures representing a set of possibly non-similar individuals linked by relations. For instance, the sentence *Eric knows Liliane* is true in the following model:

$$\text{Eric} \xrightarrow[\textit{knows}]{} \text{Liliane}$$

The central task in automated deduction is to find out whether there exists, for a given formula, a model in which it is true. This is called the *satisfiability* task. We have just seen that the answer to this question was “yes” for *Eric knows Liliane*, but what about *Everyone is tall and knows someone small* ? Even if one has an intuition of the answer in that precise case, how does one *prove* it?

Moreover, how does one get an answer for problems coming from concrete applications, where formulas and models at stake are much larger and complex? For instance, in (Suda et al., 2010), a knowledge base containing more than 10 million facts similar to the previous example is scrutinized by a first-order theorem prover. More frequently, the development of reasoners and editors, and the standardization of formats has led to the design of knowledge bases of significant size (Horrocks, 2008).

In this thesis, we are going to study satisfiability testing algorithms in certain modal languages. These tasks will often we in the complexity class *PSPACE* or beyond, and are considered more difficult that, say, satisfiability for propositional logic.

As a consequence, particular care has to be granted to carrying out this task for such languages.

This study will unfold in two different ways. On one hand, we will present decision procedures based on the tableaux method, with the motivation of leading to efficient implementations. We will put these procedures in action by presenting and evaluation an implementation, and comparing it to existing systems.

On the other hand, we will study decidability and complexity of the satisfiability task in a particular family of modal languages: modal logics with counting operators. This family can be see as a generalization of another family of modal logics called *hybrid logics*.

Let us divide the contents of this thesis in three main parts.

Tableaux for hybrid logics

We present an update of the terminating tableaux calculi introduced by Bolander and Blackburn (2007a) for three logics: $\mathcal{H}(\mathsf{D})$ interpreted on equivalence relation frames, $\mathcal{H}(\mathsf{D})$ on arbitrary models with possibly reflexive and transitive relations, and $\mathcal{H}(\mathsf{D})$ on arbitrary models with possibly reflexive, transitive and symmetric relations.

This incremental presentation enable to see which precautions need to be observed for each logic in order to preserve properties of termination, completeness and soundness of each one of the three calculi.

This updated calculus is more suitable for implementation than the original proposal by Bolander and Blackburn. Treatment of the difference modality D is done following the calculus of Kaminski and Smolka (2009b) which was the first terminating calculus handling this modality.

We also propose extensions of our calculi that do not guarantee termination. Our three extensions are role inclusions, injective and functional modalities, and the down-arrow binder $\downarrow$.

Logics with counting

We study a relatively unusual family of modal logics, from the point of view of automated deduction. The logic that interests us in particular is the basic modal logic equipped with counting operators. We distinguish these operators from *graded modalities*, also called qualified number restrictions in description logics, which count among the set of accessible worlds from a point of the model.

We start by looking at logics for which these operators have been added: first-order logic, graded modal logic, and description logics. We recall the known complexity results for their satisfiability problem. Notably, recent tight bounds on the complexity of satisfiability for first-order logic with one and two variables and counting operators give us a more accurate picture of the situation (Pratt-Hartmann, 2005, 2008).

We introduce $\mathcal{MLC}$, the basic modal logic augmented with the counting operators $\varphi \geq n$ and $\varphi \leq n$. Surprisingly, this variant has never been studied before as such. We show expressiveness results by using the appropriate bisimulation, and show the links between this logic and the hybrid logic $\mathcal{H}(:, \mathsf{E})$ by means of a translation. Using existing complexity results of other logics with counting, we obtain complexity bounds for the satisfiability problem of this logic.

Finally, we introduce a novel inference task that can be carried out with any logic with counting: the counting task. This task consists in determining, if it if it exists, the number n such that a given set of formulas logically implies that a given formula is true in n worlds. We provide an algorithm enabling to carry out this counting task for any logic with counting, provided one has a decision procedure with model building for the satisfiability problem of the considered language. This new inference task is remarkable since, contrary to many other known tasks like validity testing or instance retrieval, it can not be reduced simply to satisfiability checking.

Implementation and evaluation

Finally, we add a concrete aspect to the previous work. We present HTab (Hoffmann and Areces, 2009), an implementation of the tableaux calculi described in this thesis. We detail its architecture and the optimisations involved.

HTab carries out the task of satisfiability on formulas of the language $\mathcal{H}(\mathsf{D}, \Diamond^-)$, with the possibility to specify some relations as being reflexive, transitive or symmetric. When the given formula is satisfiable, HTab can output a model in which it is true. The prover can also test if a formula is valid, and can carry out the task of instance retrieval. This last task consists, given a theory Γ and a formula φ, in listing the nominals n of $\Gamma \cup \{\varphi\}$ such that $\Gamma \models n : \varphi$. HTab guarantees that these tasks terminate in this logic.

HTab also experimentally supports extensions like the transitive closure operators, role hierarchies, functional and injective modalities and the $\downarrow$ binder.

Then, we tackle the problem of evaluation and benchmarking. We show how we used GridTest, a benchmarking program that can run several provers against the same set of randomly generated formulas. To generate hybrid logic formulas, we used the random generator hGen (Areces and Heguiabehere, 2003). This technique has also helped us to discover bugs in HTab and sometimes also in other implementation. As a debugging technique, this approach is also called grammar-based blackbox fuzzing and has already been used for theorem provers (Brummayer and Biere, 2009). By fuzzing, we mean feeding a program with random input in order to observe bugs.

A last advantage of GridTest is that it enables us to take advantage of computer grids by linearly reducing the running time of benchmarks. This has proven valuable for the development and debugging of HTab.

Map of the thesis

We now present the contents of the chapters of this thesis.

Chapter 2 : Complexity and logic This chapter gives the basics of complexity theory defined with Turing machines. In particular we see the usual definitions of complexity classes based on time and space bounds. Then, we turn to the definitions and properties of various logics that will be useful for our investigations: modal logics, hybrid logics, description logics and first-order logics. The aim is to prepare the ground for the results later presented in the thesis.

Chapter 3 : Introduction to Tableaux Calculi This chapter gives the basics of the tableaux calculi. It starts by a brash course of this method for propositional and modal logics. In the second case, the method with prefixes is shown. Then, we recall the history of this method along with recent trends for this family of algorithms.

Chapter 4 : Prefixed Tableaux for Hybrid Logics This chapter presents tableaux decision procedures for several hybrid languages. In particular, three logics are incrementally handled: hybrid logics on models with a unique accessibility relation that is an equivalence relation, hybrid logic on arbitrary models, and hybrid logic on arbitrary models with the converse modality. For each of these three cases, the calculus proposed is proved to be adequate, that is, termination, completeness and correctness are proven. Moreover, we propose some modifications of these calculi so as to handle extensions of hybrid logic. These extensions are: role inclusion, functional and injective modalities, and the binder $\downarrow$.

Chapter 5 : Logics with counting In this chapter we consider the addition of counting operators to first-order, modal and description logics. These operators enable to express sentences of the type “at least n/at most n objects are of the kind a”. In each of the considered language, we see that the question has already been studied before, and we make a detailed state of the art. We also introduce a modal logic equipped with such operators, and study its complexity and expressive power. We highlight its connection with hybrid logic and other modal logics. Finally, we introduce a new inference task: the counting task. We give a general algorithm enabling to carry out this counting task in all logics with counting, provided one has, for the considered logic, a satisfiability decision procedure with model building.

Chapter 6 : HTab system description In this chapter we present HTab, an implementation of the tableaux algorithms presented in Chapter 4. We show how to use this prover and discuss optimizations it includes. We also evoke differences between the calculus implemented and its initial definition. Finally we evoke other similar systems in hybrid and description logics.

Chapter 7 : Benchmarks This last chapter deals with the question that follows all implementation: evaluation. This topic is presented from the point of view of benchmarks. We evoke problems related to provers evaluation, and we present the system we used to evaluate HTab, called GridTest. We also present how to run tests on parallel on machines of a grid. After showing two sample tests, we end with some perspectives for the improvement of this system.

Chapter 2

Complexity and Logic

2.1 Computability and complexity

In this section, we will present a basic toolbox to computation and complexity aimed at helping understanding the remainder of this thesis. For more information, the reader can refer to (Arora and Barak, 2009).

The intuitive and generally accepted notion of *computation* refers to the process of producing an output from a given input in a finite number of steps. The question of *computability* and *complexity* are about studying, respectively, whether a computation is possible, and whether it is hard, or resource-intensive, to carry out. In this thesis we are interested in these two problems for many logical tasks.

Among the possible tools that help referring to the notion of computation, computability and difficulty of computation, the Turing Machine (Turing, 1936) is a traditional option and is the one we are going to use.

2.1.1 Turing Machines

Turing Machines are the most common way of formalizing computation and algorithms. A Turing Machine (TM) is the formal representation of a machine able to read and write data on tapes, and able to keep track of its state by using an inner automaton. It is given an input on one of its tapes, and may output an answer on one of its tapes. It may also stop working by arriving to a special state of its automaton. It may also run infinitely on some or all of its inputs.

There are several equivalent ways to describe a TM. We choose to present a version with three tapes, the input, working and output tape. These tapes are infinite in one direction. Moreover the symbols written in the tapes cells will only be 0's and 1's. Each tape is equipped with a tape head that can read and write symbols one cell at a time.

Definition 1 (Turing Machine). *A Turing Machine is a tuple $(Q;\delta)$ where:*

- *Q is a finite set of states, containing the initial state q_{start} and the final state q_{halt}*
- *δ is the transition function: $\delta : Q \times \{0,1\}^3 \rightarrow Q \times \{0,1\}^2 \times \{L,S,R\}^3$. This function associates to every current state and every symbol read on the three tapes a symbol to*

write on the work tape and output tape, and a move of the reading heads (Left, Stop, or Right)

With this definition, it is clear that the machine's computation is divided into discrete time steps, and at each step, depending on the current automaton state and the three values read by the heads, the following actions are done: writing a bit with the working and output heads, moving each head to the left or the right, or letting it still, and moving to another state of the automaton. The machine's final answer, e.g, the result of an addition, may be read on the output tape. The automaton is initially set at q_{start}. When it enters the q_{halt} state, the machine can not make any further steps. We say in that case that the machine has halted or terminated.

This definition is not the most general definition of a TM. Indeed a Turing Machine may have any finite number of working tapes, and these may be infinite in both directions, and may hold any set of symbols, instead of just 0 and 1. However, such a general machine can be reduced into an equivalent machine as defined in Definition 1 that runs in time only polynomially larger (Arora and Barak, 2009).

Our use of the word "only" comes from the fact that a polynomial slowdown is considered as acceptable for *reducing* a problem into another one. We will come back to that claim later.

Now, let us connect the notion of a function to the notion of a TM. We also introduce the notion of running time:

Definition 2 (Computing a function and running time). *Let $f : \{0,1\}^* \to \{0,1\}^*$ and $T : \mathbb{N} \to \mathbb{N}$ be some functions, and let M be a Turing Machine. We say that M* computes f in $T(n)$-time *if for every $x \in \{0,1\}^*$, if M is initialized to the start configuration on input x, then after at most $T(|x|)$ steps it halts with $f(x)$ written on its output tape. We say that M* computes f *if it computes f in $T(n)$ time for some function $T : \mathbb{N} \to \mathbb{N}$.*

Definition 3 (Turing-computable functions). *A function $f : \{0,1\}^* \to \{0,1\}^*$ is Turing-computable if there exists a TM that stops on the input $s \in \{0,1\}*$ if and only if $f(s)$ is defined, and in that case, its output is $f(s)$.*

Those definitions extend to all functions that manipulate arbitrary strings of data, since those can always be encoded in binary.

Let us decide on a convention following which a TM answers "accept" or "reject". For instance, let us say that the first cell of the output tape contains 0 and 1 respectively for the "reject" and "accept" answers. Focussing on a TM's binary answer with regard to its inputs means focussing on a binary function. This will help us connect the notions of binary functions and *languages*.

Definition 4. *A language over an alphabet Σ is a subset of Σ^*, that is a set of words over this alphabet.*

Now, say we know about a language $\mathcal{L}$ this language may be described by a grammar, a rational expression of by a more convoluted description, e.g., "the set of positive integer numbers that are the sum of theirs proper positive divisors" (also known as perfect numbers). Now, consider the function $f : \{0,1\}^* \to \{0,1\}$ such that $f(x) = 1$ if, and only if, $x \in \mathcal{L}$. We call this function the *characteristic function* of $\mathcal{L}$.

Now, with Turing-computable functions, we know we can take the calculation of a function as a computational problem. As a consequence, we often say that a language is a problem, and that a language can be easy or difficult. This refers to the difficulty of the computation of its characteristic function. When it comes to the possibility of the computation of this function, we use the following terms:

Definition 5. *A language is* decidable *if there exists a TM that answers "accept" after finitely many steps for all word of the language and "reject" after finitely many steps for all word outside of the language.*

A language is semi-decidable *if there exists a TM that answers "accept" after finitely many steps if and only if the input belongs to the language.*

We call undecidable *any language that is not decidable.*

We use the following pieces of notation to classify languages according to these fresh notions:

Definition 6. *The class RE is the set of semi-decidable languages. RE stands for "recursively enumerable".*

The class coRE is the set of languages whose complements belong to RE.

A language that belongs to RE and coRE is said to belong to the class R. R is the set of decidable, or "recursive", languages.

The class *RE* can also be characterized by the following property:

Claim 1. *If $\mathcal{L} \in RE$, then there exists a TM that can enumerate all elements of $\mathcal{L}$.*

Now, let us see the connection between Turing Machines and the algorithms we all know. This connection has been phrased into the following statement (Goldin and Wegner, 2005):

Claim 2 (Church-Turing Thesis). *Whenever there is an effective method (algorithm) for obtaining the values of a mathematical function, the function can be computed by a TM.*

This statement gained common acceptance among mathematicians after that three independently proposed models for computation, Turing Machines, lambda-calculus and recursive functions, where proved equivalent in 1939 by Rosser (1939). It is thus believed that these models must be the right model for computation carried out by traditional computers.

What it means for us in this thesis is that all algorithms written in pseudocode, containing usual directives like variable assignments, branching and loops, are also representable as Turing Machines. Thus we will never define a TM to carry out a given computational task, but instead rely on usual pseudocode and high level explanations.

2.1.2 Complexity classes

Let us turn to the definition of difficulty of a computation. A *complexity class* is a set of functions that can be computed within a given resource. The resources we consider are time and space, both expressed in discrete steps or pieces.

Complexity classes will help up categorize problems according to their *worst-case* hardness. Consider, for instance, a function f that belongs to the class "problems that take 10,000 years". This means that for one input x at least, computing $f(x)$ takes 10,000 years. But it may be the case that for every other input y, computing $f(y)$ takes 1 second. This extreme example shows that we deal in terms of worst-case hardness, as opposed to average-case hardness.

We are going to introduce several complexity classes. Let us define the functions computable in deterministic time:

Definition 7 (Deterministic time)**.** *Let $T : \mathbb{N} \to \mathbb{N}$ be a function. We call DTIME($T(n)$) the set of all Boolean functions that are computable in $c \cdot T(n)$–time for some constant $c > 0$.*

We can now define the class P of functions computable in deterministic polynomial time:

Definition 8. $P = \bigcup_{d \geq 1} DTIME(n^d)$

In the years 1960, Cobham (1965) and Edmonds (1965) suggested that the class P may be a good formalization for efficient computation, i.e., that problems belonging to P can be considered as easy problems. One argument for this is that this class is "Turing Machine-independent", that is, the precise specification of a TM (tapes, language) does not alter the definition of this class.

We now introduce the class NP. This is the class of problems whose solutions are easily verifiable, as opposed to easily findable, as it is the case with P. For a given Turing machine M, we write $M(a,b) = x$ if it is the case that, given the concatenation of a and b as input, the Turing Machine M halts with answer x on the output tape.

Definition 9 (The class NP)**.** *A language $L \subseteq \{0,1\}$ is in NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time Turing Machine M such that for every $x \in \{0,1\}^*$,*

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$ then we call u a certificate *for x.*

So now what is polynomial, is the duration of the verification process that a word belongs to a given language. This verifications is done with a *certificate u* of size polynomial in size of x.

Why such a convoluted definition for a complexity class? Because it fits well "above" P for two reasons. First, if a language is in P then it is in NP because in that case any empty certificate will do it. Second, this class can also be defined as the problems that can be solved in polynomial time by nondeterministic Turing Machines. Those are TM who have possibly several transitions for a given state. the "n" of NP comes from this nondeterminism.

To finish with time-based complexity classes, let us introduce those that require exponential time:

Definition 10.
- *EXPTIME*$= \bigcup_{d\geq 1} DTIME(2^{n^d})$
- *2EXPTIME*$= \bigcup_{d\geq 1} DTIME(2^{2^{n^d}})$

We define the class *NEXPTIME* similarly to *NP* with regards to *P*.

Now, let us turn to space-bounded computation:

Definition 11. *Let $S : \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0,1\}^*$. We say that $L \in SPACE(s(n))$ if there is a constant c and TM M deciding L such that on every input $x \in \{0,1\}^*$, the total number of locations that are at some point non-blank during M's execution on x is at most $c \cdot s(|x|)$. (Non-blank locations in the read-only input tape do not count.)*

Let us define the class of problems that require a polynomial amount of space:

Definition 12. *PSPACE*$= \bigcup_{d\geq 1} SPACE(n^d)$

The class *NPSPACE* is defined in a similar way as *NP*, but it happens equal to *PSPACE* (Savitch, 1970).

The following inclusion of classes is known:

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq NEXPTIME \subseteq 2EXPTIME$$

Finally, to locate a problem in this inclusion of classes, we need notions of comparisons between problems. If we can adequately express a problem in terms of another one, then we may be able to transfer computational hardness results. Such an adequate expression is called a *reduction*. Reductions are handy because they prevent us from directly proving that a language requires such many resource, which can be tedious, and instead they enable us to inherit hardness properties from a problem into another one.

Definition 13 (Polynomial reduction, hardness and completeness). *A language L_1 is reducible in polynomial time to a language L_2 ($L_1 \leq_P L_2$) if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$, $x \in L_1$ if and only if $f(x) \in L_2$. A polynomial-time algorithm calculating f is called a* reduction algorithm.

Let L be a language. If every language of a complexity class C (with $P \subseteq C$) is reducible in polynomial time to L, then L is said to be C-hard.

If L is C-hard and in C, then it is said to be C-complete.

Intuitively, the notion of C-hardness means "to require at least the resources described by the class C". Completeness can be seen as "requiring exactly the resources described by the class C". This means that complete problems can be taken as examples of their complexity class.

2.1.3 Examples

We will present two well-known decidable languages, or decision problems, and give for each one a decision procedure, that is an algorithm that decides after a finite number of steps whether an element belongs to these languages.

Prime numbers

The problem of determining whether a number is prime or composite has been one of the most sought-after issues in the field of mathematics and then complexity theory. We call PRIMES the set of prime numbers.

A very simple way of testing for primality is to use the Sieve of Eratosthenes. This algorithm that finds all the prime numbers less than or equal to n consists in the following steps:

- Create a list of consecutive integers from 2 to n: $(2,3,4,\ldots,n)$.
- Initially, let p equal 2, the first prime number.
- Strike from the list all multiples of p less than or equal to n. $(2p,3p,4p,$ etc.)
- Find the first number remaining on the list after p (this number is the next prime); replace p with this number.
- Repeat steps 3 and 4 until p^2 is greater than n.
- All the remaining numbers in the list are prime.

Thus n is prime if it does belong to that list. Thus we have a decision procedure for PRIMES.

However let us add that this algorithm is far from optimal since it involves deciding primality of all numbers less than n. It involves carrying out a number of steps exponential in function of the input (assuming the number is encoded in binary).

In 1975, PRIMES was shown to be in *NP* (Pratt, 1975), and in 2002 it was shown to be in *P* (Agrawal et al., 2004).

Propositional satisfiability

In logic, satisfiability checking is the task of deciding whether a formula is satisfiable, that is, whether it can be interpreted as true in some situation. For a given language L of well-formed logical formulas, we will usually call L-SAT the language of satisfiable formulas of L. Thus, checking satisfiability of a formula is done by testing whether it belongs to the language L-SAT.

The *Davis/Putnam/Logemann/Loveland algorithm* (DPLL) (Davis and Putnam, 1960; Davis et al., 1962) is a decision procedure for the propositional satisfiability problem (written SAT) when formulas are represented in Conjunctive Normal Form (CNF). It is a recursive, backtracking-based algorithm. There exist several variants of DPLL, and we present here only a simple one.

The vocabulary necessary to understand this algorithm follows:

- a literal is a propositional variable p_i or the negation of a propositional variable written as $-p_i$.
- a clause is a disjunction of literals
- a CNF formula is a conjunction of clauses
- we write $\varphi[l]$ the CNF formula obtained by setting the literal l, that is, all clauses containing l are removed from φ and all the remaining clauses C are replaced by $C \setminus \{\neg l\}$
- when a clause becomes empty, the formula is unsatisfiable under the current assignment

DPLL aims at finding a satisfying assignment of truth values to the variables appearing in the formula. Cook (1971) showed that the problem described here, called propositional satisfiability problem, was *NP*-complete.

Each step of the DPLL algorithm is the application of the first possible rule in the following list:

- unit propagation: if a clause contains only one literal, then add this literal to the current assignment and propagate its value to all remaining clauses.
- split: pick a variable, force it to be positive. If the result is unsatisfiable, backtrack and force it to be negative.

If we write l for literals and F for CNF formulas, then the DPLL procedure can be written as the following pseudo-code:

```
DP :: Formula -> {SAT, UNSAT}
DP {} = SAT
DP F
  | {} in F = UNSAT
  | F has a unit clause {l} = DP(F[l])
  | otherwise = choose a literal l and
                 if DL(F[l]) == SAT
                  then SAT
                  else DL(F[-l])
```

As done above, the DPLL algorithm does only answer SAT or UNSAT, but it can be modified so as to return the truth assignment in case of SAT.

DPLL is in fact a family of algorithms based on this structure. Instances of DPLL algorithms might include various heuristics. However, there is little hope that it is possible to improve this worst-case bound (since it is generally believed that *P*≠*NP*), It is known that this family of algorithms may take an exponential number of steps in function of the input length for unsatisfiable formulas (Pudlák and Impagliazzo, 2000), but also for satisfiable formulas (Alekhnovich et al., 2005).

2.2 Logic

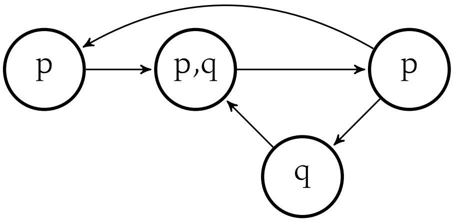
This thesis is about different logical languages, but will be mostly focussed on modal logics. We can say, in terms of complexity and expressive power, that modal logics lie between propositional and first-order logic. We do not need to give more details about propositional logic, however we are going to see in more details modal and hybrid logics, then description logics, and we will finish with first-order logic.

We will often focus on the problem of satisfiability for these languages, and give the relevant complexity results.

2.2.1 Modal logic

Modal logic designates a family of languages that are extremely flexible, while still remaining computationally well-behaved. They are adequate tools for representation and inference.

We are interested in modal logics with relational semantics, also known as Kripke semantics (Kripke, 1959, 1963). That is, we will interpret these logics on propositionally decorated graphs such as:



There are different modal languages, and we will discover a few ones by building upon what we call the *basic modal logic*. Let us describe the syntax of modal formulas with a recursive grammar. We are given a signature ⟨PROP, REL⟩, constituted of two countable and disjoint sets PROP and REL, respectively the sets of propositional symbols and relations. The grammar of the basic modal formulas is:

$$\varphi \colon = p \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \Diamond_r \varphi$$

with $p \in \mathsf{PROP}$ and $r \in \mathsf{REL}$. The following logical connectors can be defined as shortcuts: $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \quad \wedge \quad \neg\varphi_2)$, $\Box_r\varphi \equiv \neg\Diamond_r\neg\varphi$.

The Kripke semantics are given as follows. A *model* $\mathcal{M}$ is a tuple $\langle W, (R_r)_{r\in\mathsf{REL}}, V\rangle$ with:

- W a non-empty set of worlds, also called nodes or points
- R_r for each $r \in \mathsf{REL}$ a subset of $W \times W$, i.e., a binary relation on W
- $V : \mathsf{PROP} \to 2^W$, a function that associates to each propositional symbol the set of world in which it is true.

The tuple $\langle W, (R_r)_{r\in\mathsf{REL}}\rangle$ is called a *frame*. It is the underlying graph of the model, stripped of the truth values of propositional symbols. We can now define the truth of modal formulas on Kripke models, written $\mathcal{M}, w \models \varphi$ for "φ is true in model $\mathcal{M}$ at world w":

$$\begin{array}{rll} \mathcal{M}, w \models s & \text{iff} & w \in V(p), \text{ where } p \in \mathsf{PROP} \\ \mathcal{M}, w \models \neg\varphi & \text{iff} & \text{not } \mathcal{M}, w \models \varphi \\ \mathcal{M}, w \models \varphi \wedge \psi & \text{iff} & \mathcal{M}, w \models \varphi \text{ and } \mathcal{M}, w \models \psi \\ \mathcal{M}, w \models \Diamond_r \varphi & \text{iff} & \text{for some } v \in W, (w, v) \in R_r \text{ and } \mathcal{M}, v \models \varphi \end{array}$$

As truth of a modal formula in a model depends on the world of evaluation, we sometimes call *pointed model* a model $\mathcal{M}$ with one of the worlds of its domains w. A formula φ is *satisfiable* if there is a pointed model $\mathcal{M}, w$ such that $\mathcal{M}, w \models \varphi$. For a set of formulas $\Gamma \cup \{\varphi\}$ we say that $\Gamma \models \varphi$ if and only if for any model $\mathcal{M}$ and any w in its domain $\mathcal{M}, w \models \Gamma$ implies $\mathcal{M}, w \models \varphi$. This relation is sometimes called *local entailment*.

We will from time to time be lazy and only consider models with one accessibility relation, thus writing models as tuples $\langle W, R, V \rangle$. This is often safe, although there exists at least one weakly expressive modal logic for which this makes a difference in terms of complexity (Halpern and Moses, 1992).

Now we can formulate the following questions:

- given a formula φ and a pointed model $\mathcal{M}, w$, does $\mathcal{M}, w \models \varphi$ hold?
- given a formula φ, is there a pointed model $\mathcal{M}, w$ such that $\mathcal{M}, w \models \varphi$?
- given two formulas φ and ψ, is it the case that for all pointed models $\mathcal{M}, w$, $\mathcal{M}, w \models \varphi$ implies $\mathcal{M}, w \models \psi$? (also written $\varphi \models \psi$)

These questions are respectively known as model checking, satisfiability checking and logical entailment checking. The second one is the one that will focus on in this thesis.

The task of logical entailment can be reduced to satisfiability checking: if the formula $\varphi \wedge \neg\psi$ is satisfiable, then φ does not logically entail ψ. The task of model checking can not be reduced to satisfiability checking, and is in general computationally easier than satisfiability checking.

The satisfiability task is a long studied problem in computational logic. We are interested in the semantic approach, that is, exhibiting a model that satisfies a given formula, instead of having just an opaque SAT-or-UNSAT answer. We want that because a model is then a structure on which we can do queries. For instance, a model satisfying a set of formulas Γ considered as the "theory" can give us indications about whether a formula φ is implied by the theory.

Moreover, knowing about models in general may help us in the satisfiability checking task. For instance, if we know that it is possible to search only among models that have a certain shape, then this task might be simplified.

Expressive power of the basic modal logic

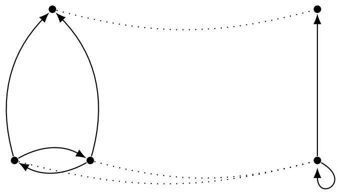
One tool to help us understand how modal logic relates to models is bisimulation. Intuitively, two pointed models are bisimilar if they comply to some properties that make them indistinguishable by the considered logic.

Definition 14. *Let $\mathcal{M} = \langle W, R, V \rangle$ and $\mathcal{M}' = \langle W', R', V' \rangle$ be two models. A* bisimulation *is a non-empty relation $Z \subseteq W \times W'$ such that when $(w, w') \in Z$:*

- *(atomic harmony) for all p, $\mathcal{M}, w \models p$ if and only if $\mathcal{M}', w' \models p$.*
- *(zig) if there is a world $v \in W$ such that $(w, w') \in R$ then there exists a world $v' \in W'$ such that $(w', v') \in R$ and Zvv'.*
- *(zag) if there is a world $v' \in W'$ such that $(w', v') \in R$ then there exists a world $v \in W$ such that $(w, v) \in R$ and Zvv'.*

Two pointed models $\mathcal{M}, w$ and $\mathcal{M}', w'$ are bisimilar *if there is a bisimulation on them containing (w, w'), and we write in that case $w \underline{\leftrightarrow} w'$.*

The two following models, although not having the same cardinality, are bisimilar:



Let us write $w \leftrightsquigarrow_{\mathcal{L}} w'$ when w and w' make true the same set of formulas in a given language $\mathcal{L}$. The following theorem is of prime importance when it comes to understanding modal models:

Theorem 5 (Bisimulation theorem for basic modal logic). *Let $\mathcal{M}$ and $\mathcal{M}$ be models. Then for every $w \in W$ and $w' \in W$, $w \underline{\leftrightarrow} w'$ implies $w \leftrightsquigarrow w'$. In other words, modal formulas are invariant under bisimulations.*

We can use bisimulations to show what is called the *tree model property*. This is obtained by the fact that *tree unravelling* is a bisimulation:

Proposition 1. *Let $\mathcal{M}$ be a model with w one of its worlds. There exist a model $\mathcal{M}' = \langle W', R', V' \rangle$ such that $\langle W', R' \rangle$ is a tree rooted at w and $\mathcal{M}, w \underline{\leftrightarrow} \mathcal{M}', w'$.*

Proof. Let W' be the set of words $ww_1w_2 \ldots w_n$ such that there exists a path $wRw_1Rw_2R \ldots Rw_n$ in $\mathcal{M}$. V' is defined as: $v' \in V'(p)$ if and only if $v' = ww_1 \ldots w_n$ and $w_n \in V(p)$.

The three conditions, atomic harmony, zig and zag can easily be verified.

The model $\mathcal{M}'$ is called the *tree unravelling* of $\mathcal{M}$ from w. □

Proposition 2 (Tree model property). *If a basic modal logic formula φ is satisfiable, then it is satisfiable in a tree-like model.*

We will see that the shape of a formula can tell us about the depth of models in which it can be satisfied.

Definition 15. *The* modal depth *D of a formula* φ *is recursively defined as:*

$$\begin{array}{rcl} D(p) & = & 0 \\ D(\Diamond\varphi) & = & D(\varphi)+1 \\ D(\Box\varphi) & = & D(\varphi)+1 \\ D(\varphi \vee \psi) & = & max\,\{D(\varphi), D(\psi)\} \\ D(\varphi \wedge \psi) & = & max\,\{D(\varphi), D(\psi)\} \\ D(\neg\psi) & = & D(\varphi) \end{array}$$

Proposition 3. *If* φ *is satisfiable and* $D(\varphi) = n$*, then* φ *is satisfiable in a model* $\mathcal{M}, w$ *whose paths from* w *without repeating worlds are of length at most* $n+1$.

Proof. By definition of $\models$, truth of subformulas of the shape $\Diamond\psi$ and $\Box\psi$ depends on truth of formulas of strictly decreasing modal depth. □

The width of a model is the maximum number of outgoing links for any of its worlds. Here again we have a restricting result:

Proposition 4. *If* φ *is satisfiable, then it is satisfiable in a model of bounded width.*

Proof. A bound can be the number of subformulas of φ the shape $\Diamond\psi$ and $\Box\psi$ (those last formulas can be nested behind negations). □

The previous three results help us understand what the basic modal logic is able to "see", and what it can not distinguish. A satisfiable formula can always be satisfied in a tree-like model of bounded depth and width. Given this obliviousness, one can hope for this logic to be computationally well-behaved, and it is indeed the case: the satisfiability problem for the basic modal logic is *PSPACE*-complete (Ladner, 1977).

Extensions of the basic modal logic

The good thing with the basic modal logic is that it is highly customisable and computationally robust. Many decidable logics can be obtained by adding new logical connectors.

Universal modality We will start with the universal modality, which corresponds to the logical connector E (Goranko and Passy, 1992). The semantics is given by:

$$\mathcal{M}, w \models \mathsf{E}\varphi \quad \text{iff} \quad \text{for some } v \in W,\ \mathcal{M}, v \models \varphi$$

The dual connector is written A, with the relation $\mathsf{A}\varphi \equiv \neg\mathsf{E}\neg\varphi$. E and A are very reminiscent of the quantifiers $\exists$ and $\forall$ of first-order logic, but they differ in the sense that they are modal, that is, they do not involve explicit quantified variables, but instead change the evaluation point of the inner formulas. Contrary to $\Diamond$ and $\Box$, E and A do not talk about the set of accessible points but the set of all points of the model.

The connectors E and A interact with the regular operators $\Diamond$ and $\Box$ so as to cause models to be possibly much bigger. In fact, while the basic modal logic can only force models of depth linear in the size of a formula, modal logic with universal modality can force exponentially deep models (see (Blackburn et al., 2001b) for an equivalent proof for Propositional Dynamic Logic). Spaan (1993) showed that the satisfiability problem for formulas of modal logic with universal modality is *EXPTIME*-complete.

Difference modality The difference modality often refers to the couple constituted of the existential difference modality D and the universal difference modality B. However the "difference modality" more often refers to the operator D (de Rijke, 1992), with semantics

$$\mathcal{M}, w \models \mathsf{D}\varphi \quad \text{iff} \quad \text{there is } v \neq w \text{ and } \mathcal{M}, v \models \varphi$$

Intuitively, $\mathsf{D}\varphi$ means "elsewhere, φ holds". In fact, Demri (1996) called the modal logic equipped with D the "logic of elsewhere", while de Rijke (1992) called it the "modal logic of inequality". We will write B the dual of D, for consistency with the notations E and A (in the literature B is often written as $\bar{\mathsf{D}}$). Intuitively, $\mathsf{B}\varphi$ means "everywhere else, φ holds", and its semantics is formally given as:

$$\mathcal{M}, w \models \mathsf{B}\varphi \quad \text{iff} \quad \text{for all } v \text{ such that } v \neq w,\ \mathcal{M}, v \models \varphi$$

So, the difference modality is almost like the existential modality. It can indeed simulate the universal modality since $\varphi \vee \mathsf{D}\varphi$ is equivalent to $\mathsf{E}\varphi$ and $\varphi \wedge \mathsf{B}\varphi$ to $\mathsf{A}\varphi$. But it is also strictly more expressive, as shown by Gargov and Goranko (1993).

Converse modality The converse modality is sometimes called the past modality, and the logic obtained is also sometimes called *tense modal logic*, as a reference to the past tense of natural languages like English. The semantics of the new modal connector $\Diamond_r^-$ (we also introduce its dual $\Box_r^-$) are defined as:

$$\mathcal{M}, w \models \Diamond_r^- \varphi \quad \text{iff} \quad \text{for some } v \in W, (v, w) \in R_r \text{ and } \mathcal{M}, v \models \varphi$$

Adding these operators to the basic modal logic does not cause any shift in complexity, that is, checking satisfiability is a *PSPACE*-complete task. This can be explained by the fact that the converse modality remains local, contrary to E and D. As a consequence, arguments that work by relating the syntactic structure of a formula with the structure of models, as it is the case for Propositions 2, 3, and 4, keep working.

Adding the converse modality to the modal logic that includes the universal modality also leaves the obtained logic as *EXPTIME*-complete.

A bit of terminology before continuing: let us write $\mathcal{M}$ the basic modal language, and specify in parenthesis extensions to this language. For instance, $\mathcal{M}(\mathsf{E}, \Diamond^-)$ is the basic modal logic extended with the universal modality and the converse modality.

2.2.2 Hybrid logic

In spite of the modularity of modal logic and the expressive power of some of its extensions, there remains the problem of the impossibility of naming worlds and expressing equality. To overcome this limitation, *nominals* were introduced in the late 1960s by Prior (1967, 1968), then independently in the 1980s by Passy and Tinchev (1985a,b). In both cases, the resulting language was highly expressive are far removed from the basic modal logic. In the 1990s, nominals were studied by Blackburn (1993), and Gargov and Goranko (1993). This marked the beginning of a growing interest into nominals and the logic obtained by using them: hybrid logic.

Introducing nominals

A hybrid logic is a logic obtained by adding *nominals* to modal logic. Nominals are propositional symbols that act as univocal names for worlds, that is, a nominal is true at exactly one world in a model. For instance, if a is a nominal, then the following formula is unsatisfiable:

$$\Diamond p \quad \wedge \quad \Diamond \neg p \quad \wedge \Box a$$

Since nominals need to be true in exactly one world, the definition of a model has to take this into account. Fix the signature $\langle \mathsf{PROP}, \mathsf{NOM}, \mathsf{REL} \rangle$ with $\mathsf{NOM} \subseteq \mathsf{PROP}$. The new constraint is that when $p \in \mathsf{NOM}$, then its valuation is a singleton set, that is, there is a single world w such that $V(p) = \{w\}$.

In addition to nominals, hybrid logic typically involves a new modal operator called the *satisfaction operator*. Given a nominal a and a formula φ, the satisfaction operator is written $a : \varphi$ (sometimes $@_a\varphi$), the intended meaning is "φ is true at the world named by a". For more convenience, we will write the satisfaction operator not only as the : symbol alone, but as the word a:, with a a placeholder nominal.

The semantics of the satisfaction operator is given by:

$$\mathcal{M}, w \models a : \varphi \quad \text{iff} \quad \mathcal{M}, v \models \varphi \text{ where } V(a) = \{v\}$$

See that in the right side of this definition, the evaluation point w is not used. This means that like E and A, a: is a global operator. Also, a: is its own dual, given that $\neg a : \varphi$ is equivalent to $a : \neg\varphi$. Let us call $\mathcal{H}(:)$ the basic modal logic with nominals and satisfaction operator.

As shown by Areces et al. (2000), adding nominals and the satisfaction operator to modal logic can have various effects in terms of complexity. On one hand, it does not cause any change of complexity for the basic modal logic, not for the modal logic with the universal modality. On the other hand, adding nominals to modal logic with the converse modality provokes a jump to *EXPTIME*-completeness. Indeed, one can simulate the universal modality with what is called a *spy point*, i.e., a world of the model that is linked to every other world. This can be done by adjoining a formula of the form:

$$s : \Box \Diamond^{-} s \quad \wedge \quad s : \Box \Box \Diamond^{-} s$$

Since the accessibility relations from this point is total, enforcing a subformula of the shape $\Box\psi$ to be true at the spy point enforces ψ to be true in all other points of the model. This technique enables to write a satisfiability-preserving translation from modal logic with the universal modality to hybrid logic with converse modality.

Coming back to the basic modal logic. From the model theoretic point of view, the addition of nominals has consequences. One visible change is that the tree model property is lost, since named worlds can have incoming links. So the evaluation point of a formula may not possibly be the root of a tree. There remains a notion of "tree submodel property" when one only considers parts of the model that does not involve named worlds.

Nominals enable expressing statements about individuals, or fixed points in time. They enable sentences like "on February 7th it rained" or "John loves Mary", instead of "it rained" or "someone loves someone". Why is the term "hybrid" used for in modal logics with nominals? Nominals enable modal logic to have two features that are traditional features of first-order logic: constants, and as a consequence, equality. This makes possible formulas like $a \wedge \Diamond a$, stating "equality between the current world and an accessible world", or to put it clearly, reflexivity of the current world. This formula could be read as $(= a) \wedge \Diamond(= a)$, but the modal nature of the language makes the equal sign implicit. Thus nominals internalise equality, turning modal logic into a hybrid of modal logic and first-order logic.

Note that the difference modality can also define nominals. If $p \wedge \mathsf{B}\neg p$ holds in a world w in a model $\mathcal{M}$, then p acts as a name of w since it can be true in no other world of $\mathcal{M}$ (Gargov and Goranko, 1993).

The down-arrow binder

Since nominals point at worlds, it was soon discovered that the ability to dynamically name worlds could lead to more interesting logics. A new hybrid operator was introduced by Goranko (1994), called the *down-arrow binder* and written $\downarrow$. It enables us to write formulas like: $\downarrow x.\varphi$, which can be understood as "after naming the current world x, φ holds". For instance, $\downarrow x.\Diamond x$ means that the current world is reflexive. This could already be enforced without $\downarrow$, as we saw in the previous paragraph, with $a \wedge \Diamond a$, but this name a would have to remain fixed, while the formula $\downarrow x.\Diamond x$ can be reused in any context, like $\Box\downarrow x.\Diamond x$ which means "all accessible worlds are reflexive".

Semantically, the ability to dynamically bind nominals involves an important difference in the evaluation of a formula in a model. While up to now, this task only involved changing of evaluation point, now because of the binder this involves changing the valuation V each time a nominal binding is done.

Let the valuation V_a^w be defined by: $V_a^w(a) = \{w\}$ and $V_a^w(b) = V(b)$ when $b \neq a$. We call that a dynamic valuation. Let the signature $\langle \mathsf{PROP}, \mathsf{NOM}, \mathsf{REL} \rangle$ be given, with $\mathsf{NOM} \subseteq \mathsf{PROP}$. A model for a formula of $\mathcal{H}(:, \downarrow)$ is a tuple $\langle W, (R_r)_{r \in \mathsf{REL}}, V \rangle$ defined as in the case of the basic hybrid logic, but with a dynamic valuation. The definition of $\models$ is thus:

$$\langle W,R,V\rangle,w \models {\downarrow}a.\varphi \quad \text{iff} \quad \langle W,R,V_a^w\rangle,w \models \varphi$$

It is understandable that the binder gives a great power to hybrid logic. The language $\mathcal{H}(:,\downarrow)$ is a reduction class of first-order logic, and is thus undecidable (Blackburn and Seligman, 1995; ten Cate, 2005). Even with a single accessibility relation, no satisfaction operator, and only nominal propositional symbols, it remains undecidable (Areces et al., 1999). In fact, $\mathcal{H}(:,\mathsf{E},\downarrow)$ is equivalent to first-order logic, since, combined with the universal modality, $\downarrow$ can define the operators $\exists$ and $\forall$, respectively with $\mathsf{E}\downarrow$ and $\mathsf{A}\downarrow$.

Still, decidable syntactic fragments of this language have been found (ten Cate and Franceschet, 2005). Although too uncontrollable to be included in the traditional modal toolbox, the binder remains typically modal since it does not break the locality of the logic. It seems in that sense more modal than a:, E and A. This is confirmed by the fact that $\mathcal{H}(\downarrow)$ is invariant under generated submodel (Areces et al., 2001a). That is, if φ is satisfied in $\mathcal{M},w$, then it is satisfied in a model made by keeping from $\mathcal{M}$ the submodel made of all worlds accessible (after an any number of steps) from w and named worlds.

The difficulty of the satisfiability task can radically change according to the modal language used, but also according to the constraints put on the model we want to obtain. For instance, testing satisfiability of a modal formula in a model made of a single irreflexive world is much easier that in an arbitrary model, since chances of saying "UNSAT" are much higher. In fact, this precise problem is equivalent to propositional SAT, which is *NP*-complete. Other constraints of the accessibility relations, like transitivity, equivalence relation or totalness, can be considered. Several such problems are studied in (Schneider, 2007).

As we said earlier, the frame of a model $\mathcal{M} = \langle W,R,V\rangle$ is the underlying graph $\langle W,R\rangle$. This graph can have properties like reflexivity, transitivity, or euclidianness. We call *frame class* the set of models whose frame have a given property. For instance the class of transitive models is the set of models such that their frame is a transitive graph. Searching for a model in a certain frame class may involve using a different algorithm than the one used to find an arbitrary model.

2.2.3 Description Logic

Let us now turn to another family of languages: Description Logics. Description Logics (DL), like modal logics, are referred to by a pluralized name since they share with modal logic a customisable aspect that make them appropriate for various real-world applications.

Description Logics are logics tailored for reasoning on taxonomies of concepts. Another name for description logics is indeed "concept languages". Two levels of knowledge representation are involved. The first one is the TBox (for Terminology Box), which contains general knowledge about the world, like "a dog is an animal", "everyone who has an animal must vaccine it". This knowledge is represented as formulas of the form $C_1 \sqsubseteq C_2$ called *general concept inclusions*. For instance

$$\begin{array}{rcl} dog & \sqsubseteq & animal \\ (human.\exists has(animal)) & \sqsubseteq & (human.\exists has(animal \sqcap vaccinated)) \\ husband & \sqsubseteq & (human \sqcap man \sqcap \exists married.human) \end{array}$$

The syntax of concepts is given as follows. We are given a signature $\langle \mathsf{CON}, \mathsf{ROL} \rangle$ made of two disjoint sets, CON being the set of concepts symbols and ROL the set of roles symbols. Complex concepts are written:

$$C: = \top \mid A \mid \neg C \mid C_1 \sqcup C_2 \mid C_1 \sqcap C_2 \mid \exists R.C \mid \forall R.C$$

with $A \in \mathsf{CON}$ and $R \in \mathsf{ROL}$.

A general concept inclusion (GCI) is a expression of the form $C \sqsubseteq D$ where C, D are complex concepts. A TBox is a finite set of GCI.

The second level of knowledge representation is the ABox (for Assertion Box), which represents individuals and properties, or predicates, attached to them. For instance, $John : husband$ is a TBox formula saying that the individual $John$ has the property of being a husband. The TBox and the ABox share the same signature. Given the previous formula and the TBox presented in the previous paragraph, one task we can do is to *saturate* the ABox with knowledge from the TBox. This may reveal inconsistencies, or enable us to then know more about the individuals of the ABox, and formulate query on them ("Does John satisfy this given property?"). In the present example, the saturated ABox could be $John$:$husband$, $John$:man, $John$:$human$, $John.\exists married.human$.

Now for the semantics. A model is a tuple $\langle D, I \rangle$ constituted of a non-empty domain D and an interpretation function I. The interpretation function I is such that:

$$\begin{array}{rcl} I(\top) & = & D \\ I(C) & \subseteq & D \\ I(\neg C) & = & D \setminus I(C) \\ I(C \sqcup D) & = & I(C) \cup I(D) \\ I(C \sqcap D) & = & I(C) \cap I(D) \\ I(R) & \subseteq & D^2 \\ I(\exists R.C) & = & \{w \in D \mid \text{ exists } v \in I(\top) \text{ s. t. } (w,v) \in I(R) \text{ and } v \in I(C)\} \\ I(\forall R.C) & = & \{w \in D \mid \text{ for all } v \in I(\top) \text{ s. t. } (w,v) \in I(R)\ ,\ v \in I(C)\} \end{array}$$

We define the relation $\models$ as follows:

$$\langle D, I \rangle \models C \sqsubseteq D \quad \text{iff} \quad I(C) \subseteq I(D)$$

and we say that $\langle D, I \rangle$ satisfies the GCI $C \sqsubseteq D$.

$\langle D, I \rangle$ is a model of a TBox T if it satisfies every GCI of T.

One computational task related to Description Logic is the task of TBox consistency checking, that is to say, to compute whether a given TBox has a model.

Now, for the ABox. Let us be given a countable set VAR of variables, disjoint from CON and REL. First, define assertional axioms, that are expressions of the form $x : C$

or $(x,y):R$ where C is a complex concept, $R \in \mathsf{ROL}$ and $x,y \in \mathsf{VAR}$. A model $\langle D,I\rangle$ is an interpretation of an ABox if $I(x) \in D$ and $(I(x),I(y)) \in I(R)$ for each of its axioms. A Knowledge Base (KB) is a pair constituted of a TBox and an ABox. A model is a model of a KB (T,A) if it is a model of its TBox T and its ABox A, and we write $\langle D,I\rangle \models (T,A)$ in that case.

The description logic we have just described is called $\mathcal{ALC}$ (Attributive concept Language with Complement) and is often taken as being the basic description logic.

Inference tasks

As Description Logics were initially motivated by concrete uses, such as ontology representation and reasoning, many inference tasks are usually considered, among which:

- Knowledge Base consistency: is there a model for a given Knowledge Base (T,A)?
- TBox consistency: is there a model for a given TBox T?
- concept subsumption: is it the case that $C \sqsubseteq D$ in a given TBox T? That is, is it the case that $T \models C \subseteq D$?
- instance checking: is it the case that $x:C$ in a given Knowledge Base? I.e, do we have $(T,A) \models x:C$?
- instance retrieval: in a given Knowledge Base (T,A), what individuals are instances of a given concept C? or what individuals are linked with a given role R? The first case is done by checking, for all individuals x in A, checking whether $(T,A) \models x:C$, the second is done by checking whether $(T,A) \models (x,y):R$, for all x,y in A.

In the case of the DL $\mathcal{ALC}$, which provides all the Boolean operators to build complex concepts, all the aforementioned inference tasks with a yes/no answer can be reduced to Knowledge Base consistency.

Extensions

One possible extension of $\mathcal{ALC}$ is the addition of nominals (Schaerf, 1994). The syntax of the language obtained, $\mathcal{ALCO}$, contains a new concept constructor $\{a_1,\ldots a_n\}$, where the names a_i belong to a given set NOM. This enable constructs such as:

$$RockFan \sqsubseteq Person \sqcap \exists hasIdol.\{Elvis\}$$

The semantics of this "one-of" construct is intuitively to be understood as "being one of the individuals listed". More formally, the interpretation I of a model $\langle D,I\rangle$ maps each nominal a to a unique individual a^I. Thus:

$$I(\{a_1,\ldots,a_n\}) \quad = \quad \{a_1^I,\ldots,a_n^I\}$$

With nominals, it seems that one can represent ABoxes directly inside of TBoxes, by appropriately using nominals in lieu of variables. Indeed, Tobies (2001a) showed that for any DL $\mathcal{L}$, satisfiability of a KB can be polynomially reduced to satisfiability of a TBox in the DL $\mathcal{LO}$.

Other extensions of ALC have been proposed, studied and used. Some of them have become usual: inverse roles, transitive roles, and number restrictions. Others are more anecdotal, like the down-arrow binder (Marx, 2002). All of this sounds very familiar to modal logicians, since Description Logics have been shown to be a syntactic variant of modal logics (Areces, 2000).

This situation of two separated formalisms, modal and description logics, while being syntactic variations one of another can be explained by their difference of focus. In the case of DL, the focus is to represent knowledge as a hierarchy of concepts, and this is adequate for many applications. As a consequence, the first-class citizen of DL is the concept, or in modal logic terms, propositional symbols. The list of inference tasks above are mostly about validity checking. In modal logics, the first-class citizen is the pointed model.

2.2.4 First-Order Logic

Let us talk about first-order logic (FOL). Considering FOL will be beneficial for several reasons. On one hand, it will help us to see modal and hybrid logics under a new light and understand how they work and what are their limitations. On the other hand, these languages inherit properties of FOL since we will see they are *fragments* of it. Thus they can also take advantage of research done in automated deduction and implementations for FOL.

First-order logic is, with propositional logic, maybe the most extensively studied logical language. It is sometimes called “classical logic”, because of the work of Frege and Dedekind in the late of 19^{th} century, who tried to use logic as the foundation of mathematics. As a consequence, the various modal logics seen in the previous sections are sometimes referred to by the term of “non-classical logics”. Given how many different logical languages we might be confronted at, we prefer not to use this vague term.

The FOL we present here is purely relational with equality, that is a FOL without function symbols but with relation = on constants symbols. Similarly to our presentation of $\mathcal{H}(:,\downarrow)$, we say that constant symbols can be dynamically bound.

The syntax of relational FOL is given according to a signature $\langle \mathsf{CON}, \mathsf{REL} \rangle$ where CON, REL are disjoint countable sets respectively of constant symbols and relational symbols. Each relational symbol R has a finite arity of $\rho(R) \leq 1$. When $\rho(R) = 1$, then R is also called a predicate. The syntax of first-order formulas is given by the following grammar:

$$F := x_1 = x_2 \mid \neg F \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid R(x_1, \ldots, x_{\rho(R)}) \mid \forall x.F \mid \exists x.F$$

for $x, x_i \in \mathsf{CON}$ and $R \in \mathsf{REL}$.

First-order formulas are interpreted on models $\langle D, I\rangle$ that are constituted of:

- a non-empty set of individuals D, called the domain.
- an interpretation function I that assigns to every constant symbol x an individual $I(x) \in D$ and to every relational symbol R a set $I(R) \subseteq D^{\rho(R)}$

When given an interpretation I and an individual $d \in D$, we define I_x^d as the function such that $I_x^d(x) = d$ and $I_x^d(z) = I(z)$. As with $\mathcal{H}(:, \downarrow)$, this is going to be useful to define truth in a model for formulas with quantifiers.

The definition of truth of a first-order formula in such a model is given as:

$$\begin{array}{rcl}
\langle D, I\rangle \models x_1 = x_2 & \text{iff} & I(x_1) = I(x_2) \\
\langle D, I\rangle \models \neg F & \text{iff} & \text{not } \langle D, I\rangle \models F \\
\langle D, I\rangle \models F_1 \wedge F_2 & \text{iff} & \langle D, I\rangle \models F_1 \text{ and } \langle D, I\rangle \models F_2 \\
\langle D, I\rangle \models F_1 \vee F_2 & \text{iff} & \langle D, I\rangle \models F_1 \text{ or } \langle D, I\rangle \models F_2 \\
\langle D, I\rangle \models R(x_1, \ldots, x_{\rho(R)}) & \text{iff} & (I(x_1), \ldots, I(x_{\rho(R)})) \in I(R) \\
\langle D, I\rangle \models \exists x.F & \text{iff} & \text{there is a } d \in D \text{ s.t. } \langle D, I_x^d\rangle \models F \\
\langle D, I\rangle \models \forall x.F & \text{iff} & \text{for all } d \in D, \langle D, I_x^d\rangle \models F
\end{array}$$

A FOL formula is satisfiable if there exists a model in which it is true, it is valid if it is true in all models.

FOL is semi-decidable, that is, one can write an algorithm that enumerates valid first-order formulas, but no such algorithm exists for non-valid formulas. As a consequence, there is no decision procedure for first-order logic. However, on one hand, automated deduction in FOL is very much explored, with lots of implementations being tested every year. On the other hand, there exist many decidable fragments of FOL[1].

The Standard Translation

Now, first-order logic without functions and with equality does not look that remote for modal logic when it comes to semantics. It does look more powerful because of the "global power" granted by the quantifiers ∃ and ∀, and the unlimited arity of relational symbols, but somehow mimicking propositional symbols of modal logics with unary predicates and accessibility relations with binary predicates seems to do the job. And it does, as shown by the following standard translation:

$$\begin{array}{rcl}
Tr_x(p) & = & P(x) \\
Tr_x(\Diamond_i \varphi) & = & \exists y. R_i(x, y) \wedge Tr_y(\varphi) \\
Tr_x(\Box_i \varphi) & = & \forall y. \neg R_i(x, y) \vee Tr_y(\varphi) \\
Tr_x(\neg \varphi) & = & \neg Tr_x(\varphi) \\
Tr_x(\varphi \wedge \psi) & = & Tr_x(\varphi) \wedge Tr_x(\psi) \\
Tr_x(\varphi \vee \psi) & = & Tr_x(\varphi) \vee Tr_x(\psi)
\end{array}$$

[1] See `http://www-mgi.informatik.rwth-aachen.de/~graedel/kalmar.pdf` for a summary.

Tr_y is defined just as Tr_x except that x and y are exchanged in its definition. We assume that neither x nor y appear in φ and ψ. The following proposition states the adequacy of this translation (Blackburn and van Benthem, 2006):

Proposition 5. *Let φ be a basic modal formula. For any modal model $\mathcal{M} = \langle W, R, V\rangle$ and world $w \in W$, $\langle W, R, V\rangle, w \models \varphi$ if and only if $\langle W, R, V_x^w\rangle \models Tr_x(\varphi)$. In other terms, φ and $Tr_x(\varphi)$ are equivalent.*

This translation can be extended so as to cover more expressive modal logics:

$$\begin{array}{rcl}
Tr_x(a) & = & x = a \\
Tr_x(a:\varphi) & = & \exists x.(a = x \wedge Tr_x(\varphi)) \\
Tr_x(\mathsf{E}\varphi) & = & \exists x.Tr_x(\varphi) \\
Tr_x(\mathsf{A}\varphi) & = & \forall x.Tr_x(\varphi) \\
Tr_x(\Diamond_i^-\varphi) & = & \exists y.R_i(y,x) \wedge Tr_y(\varphi) \\
Tr_x(\Box_i^-\varphi) & = & \forall y.\neg R_i(y,x) \vee Tr_y(\varphi) \\
Tr_x(\downarrow a.\varphi) & = & \exists a.(x = a \wedge Tr_x(\varphi))
\end{array}$$

Following ten Cate and Franceschet (2005), we can easily verify that the above translation also maintains equivalence between modal formulas of the language $\mathcal{H}(:, \downarrow, \mathsf{E}, \Diamond^-)$ and their translation.

2.3 Summing up

In this chapter we have seen how to express the cost of a computational task, in terms of time and space usage. When introducing the different logical languages that are going to be used in this thesis, we have used this notion to describe the cost of their respective satisfiability tests. In some cases, the test of satisfiability can not be made in a systematic, terminating way. This is the case, for instance, for $\mathcal{H}(\downarrow)$ and first-order logic which are undecidable.

However, we are now going to consider tableaux algorithms for hybrid logics whose satisfiability task is *NP*-complete, to begin with, and then *EXPTIME*-complete. We will start, in the next chapter, by introducing tableaux procedures for modal logics. Then in Chapter 4 we will present these calculi and prove that they are indeed decision procedures for the satisfiability problem.

Chapter 3

Introduction to tableaux procedures

Let us introduce the family of algorithms that will get us occupied for a significant part of this thesis. Although several families of decision procedures have been explored for modal logic, tableaux algorithms are maybe the most known and implemented ones.

Simply said, a tableaux algorithm is a procedure that decides satisfiability of a formula by using satisfiability of its subparts. Its is qualified of "analytic" if it only involves (possibly negated) subformulas of the input formula. This is often the case: usually a tableaux procedure does not "invent" formulas.

3.1 Crash course in propositional and modal tableaux

Let us consider propositional logic. For simplicity sake, we will consider formulas in negation normal form, i.e., formulas with the following grammar:

$$\varphi :=p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$$

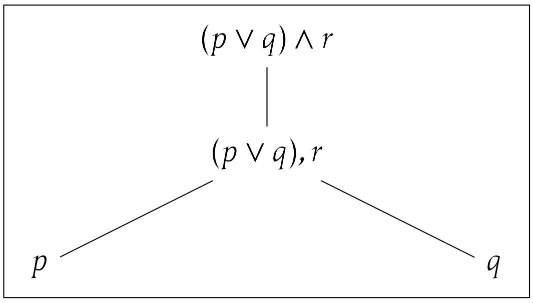
If we want to define a procedure to evaluate the satisfiability of an arbitrary propositional formula, we will need one rule per logical connector. The first one will be $(\varphi_1 \wedge \varphi_2) \Rightarrow \varphi_1, \varphi_2$. It can be understood as "if $(\varphi_1 \wedge \varphi_2)$ holds, then φ_1 and φ_2 hold".

The second rule, for the $\vee$ connector, needs to explore one of the two disjuncts first, check whether that choice leads to the conclusion that the formula is satisfiable, and if not, try the other disjunct. For this we represent our computation as a tree, whose branches represent the possible choices made for the $\vee$ connective. Thus we write $(\varphi_1 \vee \varphi_2) \Rightarrow \varphi_1 \mid \varphi_2$. The symbol $\mid$ means that the addition of φ_1 and the addition of φ_2 belong to two separate branches in an execution of the procedure.

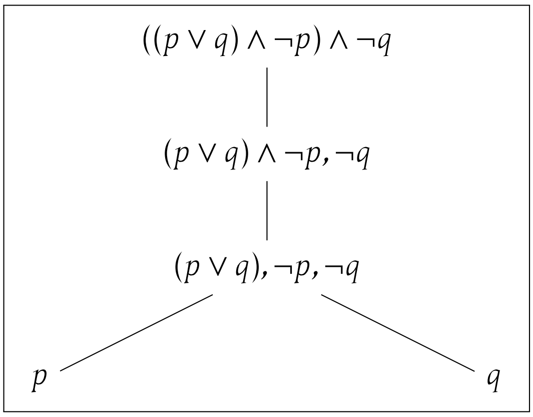
A tableau for a propositional formula φ is a tree whose nodes are sets of formulas, with its root being $\{\varphi\}$. Its edges represent rule applications, and a rule can be applied only if it adds a formula to the current set of formulas in all of the branches it creates. In the case of the rule $\vee$, a set has to have two successors.

We call a branch the union of all sets of formulas of a branch of a tableau. This represents all the constraints in a current choice of satisfiability. If both formulas p and $\neg p$ ($p \in \mathsf{PROP}$) hold in the same branch, then there is a contradiction and the current branch is said to be closed.

For instance, the formula $(p \vee q) \wedge r$ has the following tableau:



On the other hand the formula $((p \vee q) \wedge \neg p) \wedge \neg q$ has the following tableau:


$((p \vee q) \wedge \neg p) \wedge \neg q$
$(p \vee q) \wedge \neg p, \neg q$
$(p \vee q), \neg p, \neg q$
p
q

In that case both branches have a contradiction ($\neg p$ and p on the left, $\neg q$ and q on the right). We say that the tableau is *closed* and, as a consequence, we have proved that the formula is unsatisfiable.

For convenience, we often take some freedom with definition of a tableaux calculus. We assume that conjunction and disjunction symbols have variable, unlimited arity, that is to say we can write $(\varphi_1 \wedge \varphi_2 \wedge \varphi_3)$ instead of $((\varphi_1 \wedge \varphi_2) \wedge \varphi_3)$. With this assumption, a single rule application can treat all conjuncts or disjuncts at once.

In the case of modal logic, a tableaux algorithm attempts to build a model while doing the same kind of subformula exploration. Instead of only taking into account truth values of propositional symbols in one world, it does so in an arbitrary number of connected worlds.

The two new logical connectors are handled by the following two rules:

$\Diamond F$ → "create an accessible world and place F there"
$\Box F$ → "for all accessible worlds from the current world, copy F there"

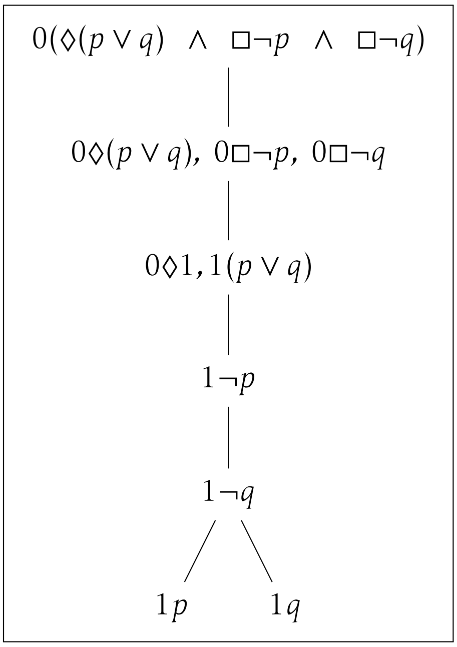
But how do we define the notions of "current world" and "accessible world"? What we need is some *localisation* of the formulas present in the tableau. Since a modal model is a set of possible worlds, we need to be able to tell for each formula in which world it holds.

So let us update our definition of a tableau with the notions of *prefixes, prefixed formulas* and *accessibility statements*. A prefixed formula is a formula of the shape $\sigma\varphi$, meaning that the formula φ is true at the world designated by the prefix σ. An accessibility statement is written as $\sigma\Diamond\tau$ and means that the world denoted by τ is accessible from the world denoted by σ. Let us summarize right away our system of now four rules:

$$\begin{array}{rcl}
\sigma(\varphi_1 \wedge \varphi_2) & \rightarrow & \sigma\varphi_1, \sigma\varphi_2 \\
\sigma(\varphi_1 \vee \varphi_2) & \rightarrow & \sigma\varphi_1 \mid \sigma\varphi_2 \\
\sigma\Diamond\varphi & \rightarrow & \sigma\Diamond\tau, \tau\varphi, \text{ for a new prefix } \tau \\
\sigma\Box\varphi, \sigma\Diamond\tau & \rightarrow & \tau\varphi
\end{array}$$

Now, the tableau is rooted at the set $\{\sigma\varphi\}$ for some prefix σ, and a contradiction occurs if both σp and $\sigma\neg p$ occur in the same branch. Let us use the natural numbers as prefixes.

Let us see what happens with the formula $\Diamond(p \vee q) \;\wedge\; \Box\neg p \;\wedge\; \Box\neg q$:


$0(\Diamond(p \vee q) \;\wedge\; \Box\neg p \;\wedge\; \Box\neg q)$
$0\Diamond(p \vee q),\ 0\Box\neg p,\ 0\Box\neg q$
$0\Diamond 1, 1(p \vee q)$
$1\neg p$
$1\neg q$
$1p$
$1q$

We have a closed tableaux since all branches are closed.

Here again, a tableaux of a formula is closed if and only if the formula is unsatisfiable. Moreover, given enough care on the application of the rule for $\Diamond$, we can prove that a procedure building a tableau always terminates. Thus, we can easily obtain a tableaux-based decision procedure for modal logic. We will discuss this in detail in the following chapters.

We will also see tableaux procedures for hybrid logics, and for various extensions of modal and hybrid logics. For now, let us first see general properties of tableaux algorithms, and let us start by a short history.

3.2 General features

Vocabulary A *tableau system* or a *tableau algorithm* contains a set of rules and a set of formulas. Each rule has its condition on when to apply, generally defined on the presence of a formula of a certain shape. The tableau system may contain additional, global constraints that may prevent the application of rules in some cases. It also defines what is a closed branch.

We call *tableau* the graph obtained from a formula by the application of a set of *rules*. This graph very often happens to be a tree, but it can also be a directed acyclic graph. We will from now on mostly care about the tree approach.

A branch of a tableau is *saturated* when it can no longer be expanded by using one of the available tableaux rules. A branch can also be open or closed. Intuitively, a closed branch is a branch known to be contradictory (for instance, containing p and $\neg p$). The precise definition of a closed branch often changes from one tableaux system to another. A closed branch is forever closed, an open branch may be found closed later after applying additional tableau rules. Only a saturated open branch is forever open.

Given a tableaux system T, let us write $T(\varphi)$ to refer to a tableau obtained by running the system T on the input formula φ.

When it comes to adequacy of a tableau calculus, three properties have to be considered: termination, soundness and completeness. While the first notion is, by now, clear, the last two need to be defined:

Definition 16 (Soundness). *A tableaux system T is* sound *if for any formula φ, if φ is satisfiable then $T(\varphi)$ is open.*

Definition 17 (Completeness). *A tableaux system T is* complete *if for any formula φ, if $T(\varphi)$ is open then φ is satisfiable.*

Soundness could be rephrased as: "the tableaux calculus finds all the satisfiable formulas". In any other computational logic context, this is in fact called completeness. This permutation of terms comes from the fact that tableaux systems were initially used to show validity of formulas, by showing unsatisfiability of their negation. The term "completeness" for tableaux is also used in the opposite way in the literature. In the rest of this thesis we will continue using these terms as defined above.

Contents of tableaux Let us consider tableau whose nodes are sets of formulas. It is not always the case that these formulas are formulas of the object language. In other words, a tableau for a formula of the language $\mathcal{L}$ may contain formulas of the language $\mathcal{L}' \neq \mathcal{L}$. This was not the case with the example for propositional logic of the previous chapter, but it was for modal logic since, e.g., $0\Diamond\varphi$ is not a modal formula but a prefixed modal formula. This is because we need to keep track of the various worlds of the model that are being built, and this is not expressible in the basic modal logic.

When moving from modal to hybrid logic, things can be different. Because the hybrid languages have nominals and the satisfaction operator, the language internalizes the labelling we have used for modal tableaux (Blackburn, 2000). It becomes possible to use only formulas of the object language, without adding another sort of symbol. Links between worlds can be represented by formulas of the shape $i : \Diamond j$. However there exist tableaux systems for hybrid logics which do use prefixes, such as (Bolander and Blackburn, 2007a) and the systems presented in the next chapter. Thus, they need to handle the notion of equality across nominals and prefixes, i.e., a prefix and a nominal can both refer to the same world.

Non-determinism The construction of a tableau is a non-deterministic process in two ways. First, there is branching non-determinism, that is, when doing a depth-first exploration of the tableau, choosing which of the two branches spawned by the ($\vee$) rule should be explored first.

This non-determinism is an issue when it comes to proving satisfiability of a formula, because one wants to find an open branch as soon as possible. When it comes to proving unsatisfiability, or to systematically explore the whole tableau, this determinism is no longer an issue since it is just a matter of reordering different pieces of computation. This is sometimes called *don't know* non-determinism.

On the other hand is the non-determinism coming from the choice of the rule application at each step of the algorithm. Indeed, there is often a choice between different rules to apply, and for a given rule, different premises formulas that can be chosen. Because of this, there can be several tableaux for a given formula. This seems problematic since we want to show adequacy of a tableaux calculus by proving that a formula is satisfiable if and only if its tableau is open. But it is often the case that the order of rule applications does not play a role in the adequacy of the calculus. This is why this is sometimes called *don't care* non-determinism.

Subformula property Tableaux algorithm work by starting from a considered formula, and taking it apart into subformulas. One natural property we would want to verify is that formulas that appear in a tableau be only subformulas of the initial formula. This is the subformula property.

Let φ be a formula of some language $\mathcal{L}$, and let $S(\varphi)$ be the *subformula and single negation closure* of φ. Proving that all formulas that appear in a tableau of φ belong to this set is a very helpful step in showing termination of a tableaux calculus.

A side-effect of this property is that the set of formulas that will appear in the tableaux is finite, and belongs to a set that is known before starting the calculus. From the point of view of implementation, this enables to have a mapping between formulas and integers, so as to represent formulas of a tableaux only as integers; this is a technique used by the hybrid prover Spartacus (**?**).

Tableaux building and caching Tableau construction is usually done in a depth-first way. That is, tableaux algorithms usually do not consider more than a single branch at the same time. This enables the algorithm to throw away previous closed branches during its course, by working in a backtracking manner.

The drawback of this method is that while the exploration of the tableau is global, no information is shared between branches. For instance, there may be sets of formulas whose satisfiability is known that could reappear in many branches of the tableau. Maintaining such information is called "caching".

Description Logic Prover (DLP) (Patel-Schneider, 1998), was one of the first implementations which used caching as an optimization. However, it is only a few years later, see for instance (Goré and Postniece, 2008) and (Goré and Nguyen, 2007), that systematic caching was taken into account at the level of the algorithm so as to obtain genuinely different systems. One big difference is that global caching enables the tableaux to be explored in any order, not necessarily depth-first.

3.3 Model building

One way to guarantee that the answer of a tableaux procedure is correct in the SAT case, is to show a model built from the found open branch, and check that the model is indeed a model of the input formula. In fact this will naturally enable to show completeness of our tableaux systems, since the model built from an open branch will prove the input formula to be satisfiable.

In the case of modal logic, extracting a model from an open branch is a relatively simple task. A saturated branch contains the frame information, i.e., what worlds exist and how are they linked, and what literal are forced to be true or false in these worlds. In that sense, a branch has more information than a model: in a given world, a propositional symbol can be known as true, known as false or unknown. In the last case, when it comes to building a model, this unknown truth value has to be chosen.

There are many applications of model building, or more directly, of models. Generally speaking, building a model of a set of formulas enables later to make queries on it. An application is the generation of referring expressions by generating the shortest formula that describes a world of the model built from a theory representing a discourse (Figueira and D.Gorín, 2010).

3.4 History

Tableaux methods evolved from Gentzen's sequent calculus. Indeed, Gentzen (1935) showed that true sequents can always be given a proof in which all formulas involved were subformulas of the input. This pivotal result opened the way to more focussed proof procedures.

Beth (1955) introduced the technique of "semantic tableau" as a method for constructing counter-examples. This technique was also independently found by Hintikka (1955a,b). In 1968, Smullyan gives the first unified and systematic exposition of semantic tableaux in his book on first-order logic (see Smullyan, 1995), calling them "analytic tableaux". Smullyan's work was extended by Fitting (1972) to the case of modal logic. Fitting gives a prefixed tableau system similar to the one given in the previous section. For a detailed story on the genesis of tableaux, one can refer to (Anellis, 1990).

The first tableau algorithm for the description logic $\mathcal{ALC}$ was proposed in 1991 by Schmidt-Schauß and Smolka (1991). Since the success of DL, starting from years mid-1990 to now, tableaux have been investigated more than ever. What sets apart the study of tableaux in description logics as opposed to before, is the stress on performance and concrete applications, that was absent from the work of Smullyan and Fitting. This is because of the applied orientation of tableaux and SAT-based deduction in DL. In the recent years, tableaux have evolved in two directions: one is to handle more and more expressive DL, and the other is to be more and more efficient in time. Attempting to list all algorithms proposed would be futile.

Let us rather focus on the development of hybrid logic tableaux. Hybrid tableaux started to be proposed while investigation on the complexity and decidability of hybrid languages was going on. As a consequence, the first tableaux proposed not always gave the right guarantees of termination.

Tzakova (1999) proposed a prefixed calculus in 1999 that covers the basic hybrid logic but also its undecidable extensions $\mathcal{H}(:, \downarrow, \forall)$ (with $\forall x.\varphi \equiv \downarrow x.\mathsf{A}\downarrow y.x : \varphi$). She proposes a restriction in order to turn the calculus of $\mathcal{H}(:)$ into a terminating one. However, as pointed by Bolander and Braüner (2006), the obtained algorithm is not terminating.

Blackburn (2000) advocates internalization of tableaux and sequent calculi for hybrid logics, by using formulas of the object language only. This can be done by directly using nominals instead of prefixes. He proposes a sound and complete tableau calculus for $\mathcal{H}(:)$, $\mathcal{H}(:, \Diamond^-)$ and $\mathcal{H}(:, \downarrow)$. Moreover, he can extend his calculus for any hybrid language whose axiom set is extended with *pure axioms*. These are formulas containing only nominals: for instance, $a \rightarrow \Diamond a$ is the pure axiom of reflexivity, $\Diamond\Diamond a \rightarrow \Diamond a$ of transitivity , and $a \rightarrow \Box\Diamond a$ of symmetry. However, no claim is made about termination of any of these calculi.

Van Eijck (2002) proposed a tableaux calculus for $\mathcal{H}(:, \mathsf{E}, \Diamond^-)$ and also for $\mathcal{H}(\downarrow)$, based on nominal substitution. This idea consists in rewriting the set of formulas contained in a branch when nominal equality is discovered. Again, termination, contrary to claim, is not guaranteed. The calculus is internalized, save from accessibility statements that remain expressed in a meta-language.

Bolander and Braüner (2006) present three hybrid tableaux calculi with a clear termination proof. The first one is a Tzakova-like prefixed calculus, the second one is a prefixed calculus based on van Eijck's substitution approach for nominal equality, and the last on is an internalized calculus based on Blackburn's proposal. In the three cases, the logic handled is $\mathcal{H}(:, \mathsf{E})$, and the calculi rely on a loop-check to ensure termination, even for $\mathcal{H}(:)$.

Bolander and Blackburn (2007a) presented two terminating calculi for $\mathcal{H}(:)$ without loop-check is introduced for the first time. The prefixed one is extended to $\mathcal{H}(:, \mathsf{E}, \Diamond^-)$ while remaining terminating, while the internalized one only handles $\mathcal{H}(:)$. In both cases equality is handled by copying formulas. The authors later proposed an extension of their internalized calculus for $\mathcal{H}(:)$ in order to handle certain frame classes in a terminating way (Bolander and Blackburn, 2007b).

A simultaneous independent proposal of terminating hybrid tableaux without loop-checks has been made by Cerrito and Mayer (2007, 2010). This time, the calculus proposed is filly internalized and, like van Eijck's, is based on nominal substitution.

Finally, Kaminski and Smolka (2007) proposed an internalised tableau calculus for $\mathcal{H}(\mathsf{E})$ based on simple type theory, and ensuring termination with a new loop-check called *pattern-based blocking*. They later extended their calculus to $\mathcal{H}(\mathsf{D})$ with transitive relations (2009b) and $\mathcal{H}(\mathsf{D}, \Diamond^-)$ (2009a). Versions handling graded modalities and role hierarchies (2009a), including graded modalities on transitive relations (2010c), and the transitive closure modality (2010b; 2010a) were also introduced. The common point of these calculi is the use of pattern-based blocking when possible, the exception being when the considered logic has the converse modality.

Chapter 4

Prefixed tableaux for Hybrid Logics

In this chapter we will present prefixed tableaux calculi for several variants of hybrid logics. These calculi are based on the one presented by Bolander and Blackburn (2007a). This original proposal has been extended to cover more expressive hybrid logics and aims at being more suitable for implementation than the original. We will also compare the present work with the one of Kaminski and Smolka (2009b,a), the first decision procedure for hybrid logic with the difference modality. The treatment of the difference modality in the present calculus is inspired from their approach. We will later discuss the differences between these two related calculi and the present one, with some remarks concerning tableaux systems for description logics.

We will consider fragments of the hybrid language $\mathcal{H}(:, \mathsf{E}, \mathsf{D}, \Diamond^-)$ defined with signature $\mathsf{Sig} = \langle \mathsf{PROP}, \mathsf{NOM}, \mathsf{REL}, \mathcal{R}, \mathcal{T}, \mathcal{S} \rangle$ where PROP is a set of ordinary propositional symbols, NOM is a set of nominals such that $\mathsf{NOM} \subseteq \mathsf{PROP}$, REL is a set of relational symbols, and $\mathcal{R}$, $\mathcal{S}$, $\mathcal{T}$ are subsets of REL. The sets PROP and REL are taken to be disjoint and are well-ordered.

The three sets $\mathcal{R}$, $\mathcal{S}$ and $\mathcal{T}$ are respectively the sets of "reflexive", "symmetric" and "transitive" relational symbols, i.e., these symbols are constrained to be interpreted as reflexive, symmetric and transitive relations. We refer to relations that do not belong to any of these sets with the term "standard relations".

The most expressive language we will consider is $\mathcal{H}(:, \mathsf{E}, \mathsf{D}, \Diamond^-)$ with full signature Sig. Notice, however, that the modality E can define satisfaction statements, since $a : \varphi$ is equivalent to $\mathsf{E}(a \wedge \varphi)$; and D can define $\mathsf{E}\varphi$ since $\mathsf{E}\varphi$ is equivalent to $\varphi \vee \mathsf{D}\varphi$. As a consequence, we will write $\mathcal{H}(\mathsf{D}, \Diamond^-)$ for $\mathcal{H}(:, \mathsf{E}, \mathsf{D}, \Diamond^-)$ and $\mathcal{H}(\mathsf{D})$ for the fragment without the converse modality $\Diamond^-$. Finally, we will write $\mathcal{H}^{S5}(\mathsf{D})$ for the language $\mathcal{H}(\mathsf{D})$ interpreted on models with a single equivalence relation.

We call Sig' the signature $\langle \mathsf{PROP}, \mathsf{NOM}, \mathsf{REL}, \mathcal{R}, \mathcal{T} \rangle$, i.e., a signature that does not specify symmetric relations.

We will consider formulas in negative normal form. Hence, the three fragments we will consider will be:

1. $\mathcal{H}^{S5}(\mathsf{D})$, given by the grammar:

$$\varphi \ ::= \ p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Diamond\varphi \mid \Box\varphi \mid a : \varphi \mid \mathsf{E}\varphi \mid \mathsf{A}\varphi \mid \mathsf{D}\varphi \mid \mathsf{B}\varphi$$

   where $p \in \mathsf{PROP}$ and $a \in \mathsf{NOM}$.

2. $\mathcal{H}(\mathsf{D})$ with standard, reflexive and transitive relations, given by the grammar:

$$\varphi \ ::= \ p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Diamond_i \varphi \mid \Box_i \varphi \mid a : \varphi \mid \mathsf{E}\varphi \mid \mathsf{A}\varphi \mid \mathsf{D}\varphi \mid \mathsf{B}\varphi$$

where $p \in \mathsf{PROP}$, $a \in \mathsf{NOM}$, $i \in \mathsf{REL}$, over the signature Sig'.

3. $\mathcal{H}(\mathsf{D}, \Diamond^-)$ with standard, reflexive, transitive and symmetric relations:

$$\varphi \ ::= \ p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Diamond_i \varphi \mid \Box_i \varphi \mid \Diamond_i^- \varphi \mid \Box_i^- \varphi \mid a : \varphi \mid \mathsf{E}\varphi \mid \mathsf{A}\varphi \mid \mathsf{D}\varphi \mid \mathsf{B}\varphi$$

where $p \in \mathsf{PROP}$, $a \in \mathsf{NOM}$ and $i \in \mathsf{REL}$, over the signature Sig.

We distinguish these three fragments since we are going to put them in correspondence with three tableaux systems that involve more and more complicated control on the application of the rules that handle subformulas of the form $\Diamond\varphi$ and $\Diamond^-\varphi$. In other words, the main difference of these systems lies in how they handle and limit the accessibility relations. Thus, the fragment with grammar:

$$\varphi \ ::= \ p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid a : \varphi \mid \mathsf{E}\varphi \mid \mathsf{A}\varphi \mid \mathsf{D}\varphi \mid \mathsf{B}\varphi$$

with $p \in \mathsf{PROP}$ and $a \in \mathsf{NOM}$, will be handled in the same manner in the three systems.

4.1 Hybrid S5 tableaux

We present a prefixed tableau calculus for the hybrid language $\mathcal{H}(\mathsf{D})$ on equivalence frames. That is, we work with a hybrid language with only one accessibility relation which is an equivalence relation. We will call the introduced tableaux calculus the *hybrid S5 calculus*.

Formulas occurring in our tableaux are *prefixed formulas* of the form $\sigma\varphi$, where φ is a formula of $\mathcal{H}(\mathsf{D})$ and σ belongs to some fixed countably infinite set of symbols PREF called *prefixes*. This set is disjoint from PROP and is well-ordered. Later, we will use the term “smallest prefix” and write $\sigma < \tau$, where σ and τ are prefixes, to refer to this well-order. The intended interpretation of a prefixed formula $\sigma\varphi$ is that σ denotes a world at which φ holds.

A tableau also contains *accessibility statements* of the form $\sigma\Diamond\tau$ where σ and τ are prefixes. The intended interpretation of $\sigma\Diamond\tau$ is that the world denoted by τ is accessible from the world denoted by σ by the accessibility relation.

Thus, the prefixed formula $\sigma\varphi$ has the same intuitive meaning as the hybrid formula $a : \varphi$, granted σ and a point to the same world. This means we could base a hybrid calculus on sets of purely hybrid formulas, as originally suggested in (Blackburn, 2000). Although it should be possible to turn the present system into an internalized one, we feel that there is an inherent distinction between nominals that are already present in the input formula and prefixes (or, in internalized calculi, nominals) created during the calculus. This distinction is that prefixes and accessibility statements are the building bricks of the frame that is being constructed, as will attest

$$\frac{\sigma(\varphi \wedge \psi)}{\sigma\varphi, \sigma\psi}\,(\wedge) \qquad \frac{\sigma(\varphi \vee \psi)}{\sigma\varphi \mid \sigma\psi}\,(\vee)$$

$$\frac{\sigma\Diamond\varphi}{\sigma\Diamond\tau, \tau\varphi}\,(\Diamond)^1 \qquad \frac{\sigma\Box\varphi, \sigma\Diamond^{eq}\tau}{\tau\varphi}\,(\Box)$$

$$\frac{\sigma\mathsf{E}\varphi}{\tau\varphi}\,(\mathsf{E})^1 \qquad \frac{\sigma\mathsf{A}\varphi}{\gamma\varphi}\,(\mathsf{A})^2$$

$$\frac{\sigma\mathsf{D}\varphi}{\sigma\neg n, \tau n, \tau\varphi}\,(\mathsf{D})^{1,3} \qquad \frac{\sigma\mathsf{B}\varphi}{\sigma n, \gamma n \mid \gamma\varphi}\,(\mathsf{B})^{2,3}$$

$$\frac{\sigma(a : \varphi)}{\sigma(a)\varphi}\,(:) \qquad \frac{\sigma\varphi}{n_\Theta(\sigma)\varphi}\,(\in)^4$$

[1] The prefix τ is new to the branch.
[2] The prefix γ is already in the branch.
[3] The nominal n is new to the branch.
[4] φ is a local formula or an accessibility statement.

Figure 4.1: Prefixed tableau calculus for $\mathcal{H}(\mathsf{D})$ on equivalence relation frames.

the definitions of models built from open branches in the three calculi. On the other hand, the propositional symbols and nominals of the input formulas will be used to define the valuation of the constructed model. Of course, there are interactions between prefixes and nominals, and we will ensure that models built from branches are well-defined with regards to the semantics of nominals.

A *tableau* is a wellfounded, finitely branching tree in which each node is a set of prefixed formulas and accessibility statements and the edges represent applications of tableau rules. Thus, rules $(\vee)$ and (B) make the tree branch into two branches. We refer to branches of tableaux as the union of all sets of formulas occurring on them, (thus this tableaux calculus is non-destructive).

When $\sigma\varphi \in \Theta$ for a branch Θ, we say that φ is true at σ on Θ, or that σ makes φ true on Θ.

A branch Θ is *closed* if $\{\sigma\varphi, \sigma\neg\varphi\} \subseteq \Theta$ for some σ and φ. Otherwise the branch is *open*. A *closed tableau* is one in which all branches are closed, and an *open tableau* is one in which at least one branch is open.

The following four definitions are required for the rules of the tableaux system presented in Figure 4.1:

Definition 18. *The relation $\sim_\Theta$ on the prefixes in a branch Θ is defined as* $\{(\sigma, \tau) \mid \sigma a, \tau a \in$

$\Theta, a \in \mathsf{NOM}\}$. *We write* $\sigma \nsim_\Theta \tau$ *when it is not the case that* $\sigma \sim_\Theta \tau$.

Definition 19. *The relation* $\Diamond^{eq}$ *on the prefixes in a branch* Θ *is the least equivalence relation containing* $\{(\sigma, \tau) \mid \sigma\Diamond\tau \in \Theta\} \bigcup \sim_\Theta$.

Definition 20. *The* nominal urfather *of a prefix* σ *in a branch* Θ, *written* $n_\Theta(\sigma)$, *is the smallest prefix* τ *in* Θ *for which* $\tau \sim_\Theta \sigma$. σ *is called a* nominal urfather *in* Θ *if* $\sigma = n_\Theta(\tau)$ *for some* τ.

Definition 21. *A* local formula *is a formula of the shape* p, $\neg p$, $\Diamond\varphi$, $\Box\varphi$, $\Diamond^-\varphi$, $\Box^-\varphi$, *with* $p \in \mathsf{PROP}$.

On a branch Θ, all the rules of the tableaux system comply with the following saturation constraints:

- Θ is open.
- A rule can only be applied if all of the successors branches it creates are proper supersets of the current branch, i.e., if every alternative conclusion adds a new formula to the branch.
- ($\Diamond$) cannot be applied to a premise $\sigma\Diamond\varphi$ if it has already been applied to $\tau\Diamond\varphi$ with $\sigma\Diamond^{eq}\tau$.
- (E) is never applied to a premise $\sigma\mathsf{E}\varphi$ if there is a prefix τ such that $\tau\varphi \in \Theta$.
- (D) is never applied to a premise $\sigma\mathsf{D}\varphi$ if there is a prefix τ such that $\tau\varphi \in \Theta$ and $\sigma \nsim_\Theta \tau$.
- (B) is never applied to a premise $\sigma\mathsf{B}\varphi$ and a prefix γ if $\sigma \sim_\Theta \gamma$.

A *saturated branch* is a branch in which no more rules can be applied that satisfy the aforementioned saturation constraints. A *saturated tableau* is one in which all branches are saturated.

We call $\mathsf{Tab}(\varphi)$ any saturated tableau whose root is the node containing exactly the following prefixed formulas:

- $\sigma_0\varphi$, with σ_0 being a fresh prefix
- $\sigma(n)n$, with $\sigma(n)$ being a fresh prefix for each nominal $n \in nom(\varphi)$.

$\sigma_0\varphi$ is called the root formula, and $\sigma_0, \sigma(n)$ (for all $n \in nom(\varphi)$) are called *root prefixes*.

Root prefixes $\sigma(n)$ are witnesses that guarantee that the nominals appearing in the input formula are true at least at one prefix.

The presence of nominals enables hybrid logic to express equality of worlds of the model. In order to handle this notion of equality, we deal with equivalence classes of prefixes (Definition 18) and representatives of these classes, namely nominal urfathers (Definition 20).

Nominal urfathers are used in the rule ($\in$), which ensures that all formulas true at prefixes of a same equivalence class end up interacting as expected. ($\in$) ensures this by always copying formulas to nominal urfathers. For instance, a branch containing $\sigma_1 p$, $\sigma_2 \neg p$, $\sigma_1 a$ and $\sigma_2 a$ with a a nominal and p a propositional symbol, will be closed thanks to the presence of ($\in$).

However ($\in$) does only need to copy local formulas (Definition 21), since the rest of the formulas either get later decomposed into formulas that may or may not be local formulas (this is the case for $\varphi \wedge \psi$, $\varphi \vee \psi$ and $\mathsf{D}\varphi$), or do not have any local influence ($\mathsf{E}\varphi$, $\mathsf{A}\varphi$, and $a : \varphi$). The last case, ($\mathsf{B}\varphi$), leads to two choices: the premise $\sigma \mathsf{B}\varphi$ provokes the addition of either two local formulas σn and γn, or to a formula $\gamma\varphi$ holding in a different prefix.

Rules ($\wedge$) and ($\vee$) are straightforward since they only deal with Boolean connectors at a given prefix.

Rule (:) directly refers to the witnesses introduced at the beginning of the calculus: when $a : \varphi$ occurs in a branch, the formula φ is copied to the prefix $\sigma(a)$ by rule (:).

Rule (E) ensures that $\mathsf{E}\varphi$ is true in σ by making φ true in an arbitrary new prefix τ. Rule (D) is very similar to (E) but it has to ensure that τ falls in a different equivalence class than σ. With that aim it introduces a new nominal n and uses it to separate the classes of τ and σ by adding σn and $\tau \neg n$ to the branch.

Rule (A) ensures that $\mathsf{A}\varphi$ is true in σ by making φ true in all prefixes of the branch. Rule (B) aims at doing this only for prefixes that are not in the equivalence class of the current prefix σ. To do so, it has to choose, for every prefix τ of the branch, either to make φ true at τ, or to make σ and τ belong to the same equivalence class. It does so by introducing a new nominal n and adding σn and τn to the branch.

When $\sigma\Diamond\varphi$ occurs in a branch, rule ($\Diamond$) creates a new prefix τ and adds it as a successor of σ adding the accessibility statement $\sigma\Diamond\tau$, and makes φ true at it. When $\sigma\Box\varphi$ holds, rule ($\Box$) sends the formula φ to all members of the "clique" to which σ belongs with respect to the accessibility relation $\Diamond$.

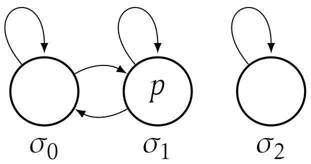
A rule is *sound* if when premise branch is satisfiable then one of its conclusions branches is satisfiable, and a tableaux system is sound if all its rules are sound. We let the reader verify that it is the case in the the present system and the two following systems of this chapter.

We mentioned earlier that the set PREF was well-ordered. Rules that introduce new prefixes on the branch – i.e., ($\Diamond$), (E), (D) – pick the smallest prefix of PREF that is not already in the branch. Similarly, rules (D) and (B) always introduce the smallest nominal of NOM that does not belong to the current branch, with the following technicality: (D) and (B) introduce nominals respectively from two alternating subsets of NOM, so that the sequence of nominals introduced by (D) is independent from the one of (B).

Examples Before showing termination and completeness of the hybrid S5 calculus, let us see a few examples of this tableaux system. We will use the symbol $\otimes$ to indicate closed branches. For each open tableau, we will show the model that should be built

$\sigma_0((\Diamond p) \wedge (\mathsf{E}\Box\neg p))$	initial node
$\sigma_0\Diamond p$	$(\wedge)$
$\sigma_0\mathsf{E}\Box\neg p$	
$\sigma_0\Diamond\sigma_1$	$(\Diamond)$
$\sigma_1 p$	
$\sigma_2\Box\neg p$	(E)
$\sigma_2\neg p$	$(\Box)$

(a)

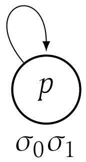


(b)

Figure 4.2: (a) Tableau for $(\Diamond p) \wedge (\mathsf{E}\Box\neg p)$ ($p \in \mathsf{PROP}$) (b) model

$\sigma_0((\mathsf{B}\neg p) \wedge (\Diamond p))$		initial node
$\sigma_0\mathsf{B}\neg p$		$(\wedge)$
$\sigma_0\Diamond p$		
$\sigma_0\Diamond\sigma_1$		$(\Diamond)$
$\sigma_1 p$		
$\sigma_0 n$	$\sigma_1\neg p$	(B)
$\sigma_1 n$	$\otimes$	
$\sigma_0 p$		$(\in)$

(a)



(b)

Figure 4.3: (a) Tableau for $(\mathsf{B}\neg p) \wedge (\Diamond p)$ ($p \in \mathsf{PROP}$) (b) model

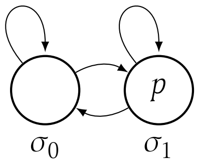
$\sigma_0((a:\neg p) \wedge b:(p \wedge \Box a))$	initial node
$\sigma_1 a$	
$\sigma_2 b$	
$\sigma_0 a:\neg p$	$(\wedge)$
$\sigma_0 b:(p \wedge \Box a)$	
$\sigma_1\neg p$	$(:)$
$\sigma_2 p \wedge \Box a$	$(:)$
$\sigma_2 p$	$(\wedge)$
$\sigma_2\Box a$	
$\sigma_2 a$	$(\Box)$
$\sigma_1 p$	$(\in)$
$\otimes$	

Figure 4.4: Tableau for $(a:\neg p) \wedge b:(p \wedge \Box a)$ ($a, b \in \mathsf{NOM}$, $p \in \mathsf{PROP}$)

from it from the information present in a saturated open branch.As a consequence, the frame of the models that follow are the equivalence relation closure of what appears in the branch. For more clarity, we will write under each world of the models

$\sigma_0 \mathsf{A}\Diamond p$	initial node
$\sigma_0 \Diamond p$	(A)
$\sigma_0 \Diamond \sigma_1$	($\Diamond$)
$\sigma_1 p$	
$\sigma_1 \Diamond p$	(A)

(a)



(b)

Figure 4.5: (a) Saturated tableau for $\mathsf{A}\Diamond p$ ($p \in \mathsf{PROP}$) (b) model

the corresponding prefix(es).

Figure 4.2 shows a formula that leads to a model with two disjoint cliques. Figure 4.3 shows how the negated difference modality is handled. It is only applied to σ_1 (since $\sigma_0 \not\sim \sigma_1$ in this branch), and leads to two branches: the left one introduces a new nominal n so that σ_0 and σ_1 are considered the same world in the model; the right one closes since σ_1 receives $\neg p$ while having p. Figure 4.4 shows ($\in$) in action and leads to a closed tableau. The last example (Figure 4.5) shows saturation of the rule ($\Diamond$) in action. Without this constraint, the tableau would be infinite.

4.1.1 Termination

To show termination, we show that there can be only finitely many prefixes on a branch, and then that the number of formulas true at any prefix in a branch is finite.

Lemma 1. *The number of equivalence classes of prefixes for the relation $\Diamond^{eq}$ in a tableau branch is finite.*

Proof. Consider the saturated tableau $\mathsf{Tab}(\varphi)$, with φ having n nominals, e distinct subformulas of the shape $\mathsf{E}\psi$ behind an even number of negations, and d distinct subformulas of the shape $\mathsf{D}\psi$ behind an even number of negations.

The number of prefixes introduced by (E) is at most e, by saturation of (E). The saturation condition of (D) is slightly different: (D) cannot be fired on a premise $\sigma\varphi$ when there is a prefix $\tau \not\sim_\Theta \sigma$ such that φ is true at τ. This condition is necessarily fulfilled after at most two applications of (D) on a same premise φ regardless of its prefix. Thus the number of prefixes introduced by (D) is at most $2d$.

On the other hand, all prefixes introduced by the ($\Diamond$) rule belong to the same equivalence class than the prefix in the premise of the rule.

Then, since there are $1+n$ root prefixes at the beginning of the tableau run, we may at most get $1+n+e+2d$ distinct equivalence classes for the relation $\Diamond^{eq}$ in a given branch. □

Lemma 2. *The number of prefixes in a tableau branch is finite.*

Proof. Let Θ be a branch of the tableau $\mathsf{Tab}(\varphi)$. Given Lemma 1, there can be only finitely many $\Diamond^{eq}$-equivalence classes in Θ. Moreover, there are only finitely many subformulas of φ of the shape $\Diamond\psi$ behind an even number of $\neg$, and thus, by saturation of $(\Diamond)$ there can be only finitely many prefixes added by $(\Diamond)$ in a branch. □

Now, let us prove the subformula property of our hybrid S5 tableaux calculus. In other words, we will show that the set of formulas true at a given prefix in a tableaux built by this calculus is finite, if we exclude all new nominals introduced by rules (D) and (B).

Definition 22. *Given a tableau branch Θ and a prefix σ, the* set of true formulas *at σ on Θ, written $T^{\Theta}(\sigma)$, is*

$$T^{\Theta}(\sigma) = \{\varphi \mid \sigma\varphi \in \Theta, \textit{ and } \varphi \textit{ is not an accessibility statement}\}.$$

Lemma 3 (Subformula Property). *Let Θ be a branch of $\mathsf{Tab}(\varphi_0)$. For any prefixed formula $\sigma\varphi$ occurring on Θ such that φ is not an accessibility relation, neither a (negated) nominal generated by (D) or (B), φ is a subformula of φ_0.*

Proof. This is easily seen by going through each rule of Figure 4.1. □

Corollary 1. *Let Θ be a branch and σ a prefix in Θ. Let $T^{\Theta}_{sub}(\sigma)$ be $T^{\Theta}(\sigma)$ without the (negated) nominals generated by (D) and (B). The set $T^{\Theta}_{sub}(\sigma)$ is finite.*

Now, it remains to be seen that in that context, there are only finitely many formulas added by (D) and (B), thus ending the proof.

Theorem 6. *The hybrid S5 calculus terminates.*

Proof. By construction, only a finite number of subformulas $\mathsf{D}\psi$ and $\mathsf{B}\psi$ can be true at some prefix in a branch. Moreover, saturation of (D) and the finite number of prefixes that occur in the branch ensures that only a finite number of nominals are introduced by rules (D) and (B). This added to Lemma 3 gives us that the set $T^{\Theta}(\sigma)$ is finite for any prefix σ in Θ. Hence $\mathsf{Tab}(\varphi)$ is finite for any formula φ. □

4.1.2 Completeness

We will now prove that the hybrid S5 tableaux calculus for $\mathcal{H}(\mathsf{D})$ is complete. For this, we will prove some properties about nominal urfathers, so that the definition of a model built from an open branch that we will see later is adequate.

Lemma 4 (Nominal Urfather Equality). *Let Θ be a saturated branch in a calculus containing $(\in)$. If $\sigma \sim_{\Theta} \tau$ then $n_{\Theta}(\sigma) = n_{\Theta}(\tau)$.*

Proof. Assume $\sigma \sim_{\Theta} \tau$. Then there is a nominal a such that $\sigma a, \tau a \in \Theta$. By saturation by the $(\in)$ rule, $n_{\Theta}(\sigma)a$ and $n_{\Theta}(\tau)a$ hold. Suppose $n_{\Theta}(\sigma) \neq n_{\Theta}(\tau)$. Without loss of generality, suppose $n_{\Theta}(\sigma) < n_{\Theta}(\tau)$. Then, a would be a nominal true at τ and $n_{\Theta}(\sigma)$, which contradicts the assumption that $n_{\Theta}(\tau)$ is the smallest prefix such that there is a nominal true at τ and $n_{\Theta}(\tau)$. □

Note that the root prefix σ_0 of a tableau branch is always a nominal urfather on that branch. More generally, any prefix σ for which $n_\Theta(\sigma) = \sigma$ is a nominal urfather on Θ. The other direction also holds, as the following lemma shows:

Lemma 5 (**Nominal Urfather Characterisation**). *Let Θ be a saturated branch in a calculus containing $(\in)$. Then σ is a nominal urfather on Θ if and only if $n_\Theta(\sigma) = \sigma$.*

Proof. Let us consider the 'only if' direction. If σ is a nominal urfather then $n_\Theta(\tau) = \sigma$ for some prefix τ. If $\tau = \sigma$ then the proof is complete. Otherwise $\sigma \sim_\Theta \tau$, by definition of n_Θ. Then, by Urfather Equality (Lemma 4), $n_\Theta(\sigma) = n_\Theta(\tau) = \sigma$. □

We are now ready to prove a correspondence between formulas in the branch and truth in the model built from it. Let us define the models we build from saturated open branches of this calculus:

Definition 23. *Given an open, saturated branch Θ of the tableau $\mathsf{Tab}(\varphi)$, we define a model $\mathcal{M}^\Theta = (W^\Theta, R^\Theta, V^\Theta)$ where:*

$$\begin{array}{rcl} W^\Theta & = & \{\sigma \mid \sigma \textit{ is a nominal urfather on } \Theta\} \\ R^\Theta & = & \{(\sigma, n_\Theta(\tau)) \mid \sigma \in W^\Theta, \tau \in \Theta \textit{ and } \sigma \Diamond^{eq} \tau\} \\ V^\Theta(p) & = & \{n_\Theta(\sigma) \mid \sigma p \textit{ occurs on } \Theta\}. \end{array}$$

Lemma 4 implies that $V^\Theta(p)$ is a singleton if p is a nominal (p is a propositional symbol or a nominal).

By construction from the relation $\Diamond^{eq}$, R is an equivalence relation.

Lemma 6. *Let Θ be a saturated open branch of a tableau $\mathsf{Tab}(\psi)$ in the hybrid S5 calculus of $\mathcal{H}(D)$. For any formula $\sigma\varphi \in \Theta$ such that $nom(\varphi) \subseteq nom(\psi)$, we have $\mathcal{M}^\Theta, n_\Theta(\sigma) \models \varphi$.*

Proof. The proof is by induction on the syntactic structure of φ.

- $\varphi = p$. By definition, $n_\Theta(\sigma) \in V^\Theta(p)$. This implies $\mathcal{M}^\Theta, n_\Theta(\sigma) \models p$.

- $\varphi = \neg p$. By saturation of $(\in)$, $n_\Theta(\sigma)\neg p \in \Theta$. Since Θ is open, $n_\Theta(\sigma)p \notin \Theta$. Thus $n_\Theta(\sigma) \notin V^\Theta(p)$, which implies $\mathcal{M}^\Theta, n_\Theta(\sigma) \models \neg p$.

- $\varphi = \psi \wedge \chi$ and $\varphi = \neg(\psi \wedge \chi)$ are trivial, by application of the corresponding tableau rules and the induction hypothesis.

- $\varphi = a : \psi$. By closure under rules (:) and $(\in)$, Θ must also contain $\tau\psi$, with τ being the smallest prefix such that τa occurs. Induction hypothesis gives us $\mathcal{M}^\Theta, n_\Theta(\tau) \models \psi$. By Urfather Characterisation (Lemma 5), $n_\Theta(\tau) = \tau$, and since $\tau a \in \Theta$, we get $V^\Theta(a) = \{\tau\}$. Thus, as we have $\mathcal{M}^\Theta, \tau \models \psi$ and $\mathcal{M}^\Theta, \tau \models a$, we have that $\mathcal{M}^\Theta, n_\Theta(\sigma) \models a : \psi$, as needed.

- $\varphi = \mathsf{E}\psi$. Closure under the (E) rule implies that Θ must also contain a formula $\tau\psi$ for some prefix τ. Induction hypothesis then gives us $\mathcal{M}^\Theta, n_\Theta(\tau) \models \psi$, thus $\mathcal{M}^\Theta, n_\Theta(\sigma) \models \mathsf{E}\psi$.

- $\varphi = \mathsf{A}\psi$. Choose an arbitrary element τ in W^Θ. By closure under the (A) rule we have that $\tau\psi$ occurs on Θ. Induction hypothesis gives us $\mathcal{M}^\Theta, n_\Theta(\tau) \models \psi$. By Urfather Characterisation (Lemma 5), we have $n_\Theta(\tau) = \tau$, thus $\mathcal{M}^\Theta, \tau \models \psi$ as required.

- $\varphi = \mathsf{D}\psi$. Closure under the (D) rule implies that Θ also contains $\sigma\neg n$, τn and $\tau\psi$. As $\sigma\neg n$ and τn occur, by saturation of $(\in)$ we have $n_\Theta(\sigma)\neg n$ and $n_\Theta(\tau)n$, so, as the branch is open, $n_\Theta(\sigma) \neq n_\Theta(\tau)$. Moreover, as $\tau\psi \in \Theta$, then by induction hypothesis, $\mathcal{M}^\Theta, n_\Theta(\tau) \models \psi$. With $n_\Theta(\sigma) \neq n_\Theta(\tau)$, this means we have $\mathcal{M}^\Theta, n_\Theta(\sigma) \models \mathsf{D}\psi$.

- $\varphi = \mathsf{B}\psi$. If there is no world $\tau \neq n_\Theta(\sigma)$ then this trivially holds. Otherwise, let τ be such a world. By saturation of (B), either the formulas σn and τn are in Θ, or $\tau\psi$ is. In the first case, $\sigma \sim_\Theta \tau$, which implies by Urfather Equality (Lemma 4) that $n_\Theta(\sigma) = n_\Theta(\tau)$. Thus, by Urfather Characterisation (Lemma 5), $n_\Theta(\sigma) = \tau$, which is a contradiction. Now assume $\tau\psi \in \Theta$. Then, by induction hypothesis and Urfather Characterisation (Lemma 5), $\mathcal{M}^\Theta, \tau \models \psi$, which is what we needed.

- $\varphi = \Diamond\psi$. By saturation of $(\in)$ and $(\Diamond)$, we have:

  $$n_\Theta(\sigma)\Diamond\tau,\ \tau\psi\ \in \Theta$$

  Then, by definition of R^Θ and induction hypothesis:

  $$(n_\Theta(\sigma), n_\Theta(\tau)) \in R^\Theta \quad \text{and} \quad \mathcal{M}^\Theta, n_\Theta(\tau) \models \psi$$

  Combining this, we obtain $\mathcal{M}^\Theta, n_\Theta(\sigma) \models \Diamond\psi$, as required.

- $\varphi = \Box\psi$. If there is no world σ_1 such that $(n_\Theta(\sigma), \sigma_1) \in R^\Theta$ then this holds trivially. Otherwise, let σ_1 be such that $(n_\Theta(\sigma), \sigma_1) \in R^\Theta$.

  By definition of R^Θ there is a prefix τ_1 such that $\sigma_1 = n_\Theta(\tau_1)$ and $n_\Theta(\sigma)\Diamond\tau_1 \in \Theta$. Then by saturation of $(\in)$, $n_\Theta(\sigma)\Box\psi \in \Theta$, and by closure under $(\neg\Diamond)$, $\tau_1\psi \in \Theta$. Induction hypothesis entails $\mathcal{M}^\Theta, n_\Theta(\tau_1) \models \psi$, i.e., $\mathcal{M}^\Theta, \sigma_1 \models \psi$. From this it follows that $\mathcal{M}^\Theta, n_\Theta(\sigma) \models \Box\psi$.

□

Theorem 7. *The hybrid S5 calculus of $\mathcal{H}(D)$ is complete.*

Proof. Let Θ be a saturated open branch of the tableau $\mathsf{Tab}(\varphi)$. Since $\sigma_0\varphi \in \Theta$, by Lemma 6 we get that φ is satisfiable. □

Note that now, we not only have a decision procedure for the language $\mathcal{H}^{S5}(\mathsf{D})$, but also a *model building* procedure, since we proved we can always build a model from a saturated open branch.

4.1.3 Discussion

We would like now to discuss some of the similarities and differences between the calculus of this section and related work. In particular we will discuss the work of Bolander and Blackburn, and Kaminski and Smolka on hybrid tableaux calculi. Bolander and Blackburn (2007a) introduced the first terminating tableau system for the basic hybrid logic $\mathcal{H}(:)$. For this language, both a prefixed and an internalised calculus were introduced. Moreover, they introduced a prefixed calculus for $\mathcal{H}(:, \mathsf{E}, \Diamond^-)$. Kaminski and Smolka (2009b) introduced an internalised calculus for $\mathcal{H}(\mathsf{D})$ with reflexive and transitive relations, and later extended it so as to handle the hybrid logic $\mathcal{H}(\mathsf{D}, \Diamond^-)$ (2009a).

Handling equivalence classes of prefixes Bolander and Blackburn used the following (Id) rule to handle equivalence classes for $\mathcal{H}(:, \mathsf{E})$:

$$\frac{\sigma a, \tau a, \tau\varphi}{\sigma\varphi}\,(Id)$$

The (Id) rule is an unrestricted version of the $(\in)$ rule. It copies all formulas of an equivalence class to all prefixes of the same equivalence class. This way of handling classes is correct but costly, as it turns out that $(\in)$ alone suffices. The approach of $(\in)$, where information is only copied to the representative prefix of an equivalence class, is equivalent to the classic disjoint-set forest approach to solve the union-find problem (Cormen et al., 2001).

The calculus of Kaminski and Smolka handles equivalence classes by making rules depend on the equational congruence of a branch. That is, the closure of a branch obtained by rewriting every formula and every accessibility statement by replacing every nominal by any other nominal of its equivalence class. For instance in (Kaminski and Smolka, 2009a), negation is handled by two rules as follows (side conditions are written on the right of each rule):

$$\frac{x \neq y}{\perp}\; x \sim_A y \qquad\qquad \frac{(\dot\neg p)x}{\perp}\; px \in \tilde{A}$$

with $\tilde{A}$ being the equational congruence of a branch A and $\sim_A$ the least equivalence relation on the nominals of a branch.

In our case, we make explicit the handling of equivalence classes by coping the adequate formulas to representative prefixes, and letting the other rules deal directly with the prefixed formulas present in the branch. In this way our tableau algorithm directly handles equivalence classes. Instantiating Kaminski and Smolka's approach with a disjoint-set forest should yield a very similar system.

Kaminski and Smolka's calculus and the difference modality Kaminski and Smolka (2009b) presented the first decision procedure for hybrid logic with arbitrary relations and the difference modality. Their calculus is internalised, for it is expressed in simple type theory, and equality and disequality are represented natively in their formalism with the symbols = and ≠. In contrast, our calculus is prefixed, and we use new nominals to enforce equality and disequality, respectively needed by the rules (B) and (D), which we adapted from their work.

Demri's "Logic of Elsewhere" calculus Demri (1996) introduced a tableau calculus for propositional logic with the difference modality without accessibility relations, universal modality nor nominals. He moreover shows that this language is strictly more expressive than modal logic on equivalence relation frames. However, both languages are not combined. Later work by Balbiani and Demri (1997) combines the difference modality with standard accessibility relations, although the calculus presented does not terminate on all input. It is only in (Kaminski and Smolka, 2009b) that a tableaux system for modal logic with the difference modality was introduced.

Complexity of $\mathcal{H}^{S5}(\mathsf{D})$ The satisfiability problem for $\mathcal{H}(:, \mathsf{E})$ on equivalence relation frames is known to be *NP*-complete (Schneider, 2007) if only a single accessibility relation is involved, thus this problem is simpler than for standard accessibility relations. Indeed, when evaluated on an equivalence relation frame, any nesting of modalities can be replaced by the last one in the string of operators, giving an equivalent formula. For instance, $\Diamond\Box\Box\psi$ is equivalent to $\Box\psi$.

Theorem 8. *$\mathcal{H}^{S5}(D)$ is NP-complete.*

Proof. *NP*-hardness comes from the fact that $\mathcal{H}^{S5}(\mathsf{D})$ is a conservative extension of $\mathcal{H}^{S5}(\mathsf{E})$, which is known to be *NP*-complete (Schneider, 2007).

Showing that $\mathcal{H}^{S5}(\mathsf{D})$ is in *NP* can be done by exhibiting polynomial-size certificates that a formula is satisfiable.

For a satisfiable formula φ, open branches of $\mathsf{Tab}(\varphi)$ contain at most $1+n+e+2d \leq 4|\varphi|$ equivalence classes (proof of Lemma 1) and at most $|\varphi|$ formulas of the form $\Diamond\psi$, that can be expanded in each class. So models built from such open branches have a size bounded by $4|\varphi|^2$, and are adequate certificates since model checking can be done in deterministic polynomial time. □

Extension to several S5 relations However, it is not possible to extend adequately this tableau system so as to handle several accessibility relations without introducing restrictions (or loop-check) on the application of the ($\Diamond$) rule. One can check that Lemma 1 no longer holds with multiple accessibility relations (each alternation $\Diamond_i\Diamond_j$ can create a new equivalence class of prefixes for $\Diamond_j^{eq}$.)

It is indeed known that satisfiability with two S5 relations is *PSPACE*-complete (Halpern and Moses, 1992). Let us write $\mathcal{H}_2^{S5}(\mathsf{D})$ for the logic $\mathcal{H}^{S5}(\mathsf{D})$ with two S5 relations. We conjecture it is *EXPTIME*-complete, however for now we have to content ourselves with the following result:

Theorem 9. *$\mathcal{H}_2^{S5}(D)$ is EXPTIME-hard.*

$$\frac{\sigma(\varphi\wedge\psi)}{\sigma\varphi,\sigma\psi}\,(\wedge) \qquad \frac{\sigma(\varphi\vee\psi)}{\sigma\varphi\mid\sigma\psi}\,(\vee)$$

$$\frac{\sigma\Diamond_i\varphi}{\sigma\Diamond_i\tau,\tau\varphi}\,(\Diamond)^1 \qquad \frac{\sigma\Box_i\varphi,\sigma\Diamond_i\tau}{\tau\varphi}\,(\Box)$$

$$\frac{\sigma\mathsf{E}\varphi}{\tau\varphi}\,(\mathsf{E})^1 \qquad \frac{\sigma\mathsf{A}\varphi}{\gamma\varphi}\,(\mathsf{A})^2$$

$$\frac{\sigma\mathsf{D}\varphi}{\sigma\neg n,\tau n,\tau\varphi}\,(\mathsf{D})^{1,3} \qquad \frac{\sigma\mathsf{B}\varphi}{\sigma n,\gamma n\mid\gamma\varphi}\,(\mathsf{B})^{2,3}$$

$$\frac{\sigma(a:\varphi)}{\sigma(a)\varphi}\,(:) \qquad \frac{\sigma\varphi}{n_\Theta(\sigma)\varphi}\,(\in)^4$$

$$\frac{\sigma\Box_i\varphi}{\sigma\varphi}\,(re), i\in\mathcal{R} \qquad \frac{\sigma_0\Box_i\varphi,\sigma_0\Diamond_i\sigma_1}{\sigma_1\Box_i\varphi}\,(tr), i\in\mathcal{T}$$

[1] The prefix τ is new to the branch.
[2] The prefix γ is already in the branch.
[3] The nominal n is new to the branch.
[4] φ is a local formula or an accessibility statement.

Figure 4.6: Rules for $\mathcal{H}(\mathsf{D})$ with reflexive and transitive relations.

Proof. $\mathcal{H}(\mathsf{D})$ with one standard accessibility relation can be encoded into $\mathcal{H}_2^{S5}(\mathsf{D})$ by replacing $\Diamond$ by $\Diamond_1\Diamond_2$ as in (Halpern and Moses, 1992). Since this is obviously a poly-time reduction, $\mathcal{H}_2^{S5}(\mathsf{D})$ is *EXPTIME*-hard. □

4.2 Tableaux for $\mathcal{H}(\mathsf{D})$ with reflexive and transitive modalities

We now move to a more elaborated tableaux system. The main difference with the previous one is that we no longer consider satisfiability of a hybrid formula on an equivalence relation frame, but on a frame with several standard accessibility relations. Moreover, some accessibility relations can be specified as reflexive or transitive.

Figure 4.6 presents the rules needed to handle the hybrid language $\mathcal{H}(\mathsf{D})$ with reflexive and transitive modalities. We call this system *the calculus of $\mathcal{H}(D)$*.

At the level of rules, the differences with the previous calculus are that $(\Diamond)$ and $(\neg\Diamond)$ now work with several accessibility relations, and that $(\neg\Diamond)$ sends formulas only to immediately successor prefixes. Moreover, rules (re) and (tr) are introduced to ensure, respectively, that the reflexive and transitive closure of accessibility relations in a branch are computed so as to ensure the specified constraints when building the model induced by the tableau branch.

The saturation constraints of the rules of this system are the same than for the previous calculus, except for ($\Diamond$):

- ($\Diamond$) can not be applied to a premise $\sigma\varphi$ on Θ if it has already been applied to $\tau\varphi$ with $\sigma \sim_\Theta \tau$

See that this saturation condition is weaker that the one of the hybrid S5 calculi, which means that now, formulas such as $\mathsf{A}\Diamond p$ no longer terminate: the calculus creates an infinite strand of prefixes because of the $\Diamond p$ subformula added to every one of them.

To solve this problem, we introduce an extra constraint on the calculus to prevent such infinite strands of similar worlds to occur. We are going to divide prefixes in two categories: those who "can generate new prefixes with ($\Diamond$)" and those who cannot. During the run of the calculus, prefixes of a branch can pass from one category to the other. Ultimately, we will see that there can be only a finite number of prefixes belonging to the first category, which will imply termination of the calculus. The next definition is necessary to define these categories:

Definition 24. *For a prefix σ, let $L^\Theta(\sigma)$ be the set of formulas true at $n_\Theta(\sigma)$, of the shape $\Diamond\varphi$, $\Diamond^-\varphi$, $\Box\varphi$, $\Box^-\varphi$, p and $\neg p$, with p being a propositional symbol or a nominal not introduced by the rule* (B). *We call these formulas* model-relevant local formulas.

A prefix σ will belong to the blocked category when its L^Θ set is included in the L^Θ set of a smaller prefix. If σ is a blocked prefix, diamond expansions of formulas of the form $\sigma\Diamond\varphi$ are forbidden.

Definition 25. *The* inclusion urfather *of a prefix σ in a branch Θ, written $i_\Theta(\sigma)$, is the smallest prefix τ for which: $L^\Theta(\sigma) \subseteq L^\Theta(\tau)$. A prefix σ is called an* inclusion urfather *in Θ if $\sigma = i_\Theta(\tau)$ for some prefix τ.*

The following condition can be seen as an extra saturation condition on the rule ($\Diamond$), however it is different in the sense that it is a *global* condition on the branch:

Definition 26. *(***Loop-check** *($\mathcal{I}$)) The rule* ($\Diamond$) *is only applied to a formula $\sigma\varphi$ on a branch if σ is an inclusion urfather on that branch.*

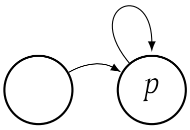
One consequence of the loop-check $\mathcal{I}$ is that a formula $\Diamond\varphi$ has to be copied to the nominal urfather of its prefix before being expanded. In the section that follows, we will see how this restriction ensure termination. Before that, let us see a few example runs of this calculus.

Example Before showing adequacy of this calculus, let us see an example. With the present calculus, we no longer have to take the equivalence closure of relations of the model. We also can have several different relation symbols, although for the sake of simplicity we will only use a single one. Most interestingly, we need to do something with blocked prefixes: if a saturated open branch contains some prefix σ with a non-expanded formula $\sigma\Diamond\varphi$, we can not directly represent this prefix in a model as before. Intuitively, such a prefix will be represented in the model by its inclusion urfather. Thus we should also rewire all links pointing to σ onto its inclusion urfather.

Figure 4.7 shows an example with the formula $\mathsf{A}\Diamond p$ which, without the loop-check and only the saturation condition of ($\Diamond$), would provoke an infinite tableau. In the saturated open branch, σ_2 is blocked since σ_1 is its inclusion urfather. As a consequence, there are only two worlds in the model, and the outgoing link from σ_1 is rewired on σ_1 itself, which is safe since σ_1 has at least as much "local" information as σ_2 (the set of local formulas $\{p, \Diamond p\}$).

$\sigma_0\mathsf{A}\Diamond p$	initial node
$\sigma_0\Diamond p$	(A)
$\sigma_0\Diamond\sigma_1$	($\Diamond$)
$\sigma_1 p$	
$\sigma_1\Diamond p$	(A)
$\sigma_1\Diamond\sigma_2$	($\Diamond$)
$\sigma_2 p$	
$\sigma_2\Diamond p$	(A)

(a)



(b)

Figure 4.7: (a) Saturated tableau for $\mathsf{A}\Diamond p$ ($p \in \mathsf{PROP}$) (b) model

4.2.1 Termination

Again, termination is proven showing that both the number of prefixes in a branch is finite and the size of T^Θ are finite. The first part is different since it now relies on the loop-check: it is, basically, an application of the pigeonhole principle with respect to all possible sets L^Θ.

Lemma 7. *Let Θ be a branch in the calculus of $\mathcal{H}(D)$ with finitely many prefixes in it, and σ a prefix occurring in it. $T^\Theta(\sigma)$ is finite.*

Proof. Same argument than in the proof of Theorem 6. □

Lemma 8. *Let* Tab(φ) *be a tableau in the calculus of $\mathcal{H}(D)$. There are finitely many possible sets $L^\Theta(\sigma)$ for any branch Θ and prefix σ in* Tab(φ).

Proof. Only a finite number of formulas of the shape $\mathsf{D}\varphi$ can appear in a branch, and (D) can be applied at most twice for each one of them as premise. Let N be the set of nominals that can be introduced by (D) in this calculus, that is, N is the set of the smallest $2d$ nominals of NOM not being subformulas of φ. This and Corollary 1 prove that for any prefix σ, the set $L^{\Theta}(\sigma)$ is a subset of the finite set $U \cup N$, where U is the set of all quasi-subformulas of φ and N is defined as previously. □

Theorem 10. *Any tableau in the calculus of $\mathcal{H}(\mathsf{D})$ constructed under restriction $(\mathcal{I})$ is finite.*

Proof. The prefixes present in a branch are either root prefixes, prefixes introduced by (E) and (D), or prefixes introduced by ($\Diamond$). We already know that there are finitely many root prefixes. Moreover, as only a finite number of subformulas of the input formula are of the shape $\mathsf{E}\varphi$ or $\mathsf{D}\varphi$, the saturation condition of (E) and (D) ensure that only a finite number of prefixes can be generated by these rules. Let us now consider the number of diamond expansions:

- because of Lemma 8 and the loop-check $(\mathcal{I})$, the maximal number of equivalence classes in which prefixes are allowed to expand diamonds is 2^N, where N is the number of model-relevant formulas.

- because of saturation of ($\Diamond$), the maximal number of diamond expansions in a given equivalence class is M, the number of subformulas of the input formula of the shape $\Diamond\varphi$.

Thus the number of prefixes generated by ($\Diamond$) in a branch is bounded by $M \times 2^N$.

As there can only be finitely many prefixes in a branch, the result follows by Lemma 7. □

4.2.2 Completeness

We will now prove that the calculus of $\mathcal{H}(\mathsf{D})$ is complete. For this, we need a certain amount of properties about inclusion urfathers, so that a saturated open branch has the desirable properties. We already showed properties for nominal urfathers in the hybrid S5 calculus, and these also hold for the present tableaux system. The first result we will establish enables us to claim, under certain conditions, that nominal and inclusion urfather are the same:

Lemma 9. *Let Θ be a saturated branch in a calculus containing ($\in$). If σ is a prefix making at least one model-relevant nominal true on Θ then the nominal urfather and the inclusion urfather of σ coincide.*

Proof. Assume $\sigma a \in \Theta$, with a being a model-relevant nominal. We need to prove $n_{\Theta}(\sigma) = i_{\Theta}(\sigma)$. Closure under the ($\in$) rule gives us that $n_{\Theta}(\sigma)a \in \Theta$, so $a \in L^{\Theta}(\sigma)$. Let $\tau = i_{\Theta}(\sigma)$. By definition, τ is the smallest prefix such that $L_{\Theta}(\sigma) \subseteq L_{\Theta}(\tau)$, so $a \in L_{\Theta}(\tau)$. Hence $n_{\Theta}(\tau)a \in \Theta$.

Assume $\tau \neq n_{\Theta}(\sigma)$. The case $\tau > n_{\Theta}(\sigma)$ is impossible, because then $n_{\Theta}(\sigma)$ would be a candidate inclusion urfather of σ smaller than τ, by closure under ($\in$). So let us assume $\tau < n_\Theta(\sigma)$. As $n_\Theta(\tau) \leq \tau$, we have $n_\Theta(\tau) < n_\Theta(\sigma)$, but since $n_\Theta(\tau)$ makes the nominal a true on Θ, this contradicts the fact that $n_\Theta(\sigma)$ is the nominal urfather of σ. □

We have proved two basic properties for nominal urfathers: Nominal Urfather Equality (Lemma 4) and Nominal Urfather Characterisation (Lemma 5). We are going to see that these properties also hold for inclusion urfathers.

Lemma 10 (Inclusion Urfather Equality). *Let Θ be a saturated branch in a calculus containing ($\in$). If $\sigma \sim_\Theta \tau$, then $i_\Theta(\sigma) = i_\Theta(\tau)$.*

Proof. σ and τ have the same nominal urfather (Lemma 4), thus they have the same inclusion urfather. □

Lemma 11 (Inclusion Urfather Characterisation). *Let Θ be a saturated branch in a calculus containing ($\in$). Then σ is an inclusion urfather on Θ if and only if $i_\Theta(\sigma) = \sigma$.*

Proof. For the "only if" direction, suppose σ is an inclusion urfather, i.e., there exists a prefix τ such that $\sigma = i_\Theta(\tau)$. Let us show that $i_\Theta(\sigma) = \sigma$. By definition, σ is the smallest prefix such that $L^\Theta(\tau) \subseteq L^\Theta(\sigma)$ for a prefix τ. Suppose that there is a prefix $\gamma = i_\Theta(\sigma)$ and $\gamma < \sigma$. Therefore, $L^\Theta(\tau) \subseteq L^\Theta(\sigma) \subseteq L^\Theta(\gamma)$, which contradicts the fact that σ is the inclusion urfather of τ. □

And a third property is going to be useful:

Lemma 12. *Given a saturated branch Θ in a calculus containing ($\in$) and a prefix σ, $n_\Theta(i_\Theta(\sigma)) = i_\Theta(\sigma)$.*

Proof. Let $\tau = i_\Theta(\sigma)$. Assume $n_\Theta(\tau) \neq \tau$. Necessarily, $n_\Theta(\tau) < \tau$. Since by nominal Urfather Characterisation (Lemma 5), $n_\Theta(n_\Theta(\tau)) = n_\Theta(\tau)$, and by saturation by ($\in$), $n_\Theta(\tau)$ is also a candidate to be the inclusion urfather of σ, and since it is smaller than τ, we have a contradiction. □

Lemma 13 (Inclusion Urfather Closure). *Let Θ be a saturated branch in a calculus containing ($\in$). If Θ contains $\sigma\varphi$ with φ a model-relevant local formula, then Θ contains $i_\Theta(\sigma)\varphi$.*

Proof. By saturation of ($\in$) and definition of inclusion urfather, $n_\Theta(i_\Theta(\sigma))\varphi \in \Theta$ which gives us $i_\Theta\varphi \in \Theta$ by Lemma 12. □

We can now define how to build a model based on the inclusion urfathers of a saturated open branch:

Definition 27. *Given an open, saturated branch Θ with root $\sigma_0\varphi_0$, we define the model $\mathcal{M}^\Theta = (W^\Theta, (R_i^\Theta)_{i<n}, V^\Theta)$ with:*

$$\begin{aligned} W^\Theta &= \{\sigma \mid \sigma \textit{ is an inclusion urfather on } \Theta\} \\ R_i^\Theta &= \{(\sigma, i_\Theta(\tau)) \mid \sigma \in W^\Theta \textit{ and } \sigma\Diamond_i\tau \textit{ occurs on } \Theta\} \\ V^\Theta(p) &= \{i_\Theta(\sigma) \mid \sigma p \textit{ occurs on } \Theta\}. \end{aligned}$$

Again, p is a propositional symbol or a nominal, and Lemma 10 implies that $V^{\Theta}(a)$ is a singleton for any nominal a.

We moreover define the model $\mathcal{M}^{\Theta}_{*}$ as $\mathcal{M}^{\Theta}$ in which the missing links for reflexive and transitive relations have been added. For every relation R_i in $\mathcal{M}^{\Theta}$, we write R_{i*} its reflexive closure when $i \in \mathcal{R}$, its transitive closure when $i \in \mathcal{T}$ and its reflexive-transitive closure when $i \in \mathcal{R} \cap \mathcal{T}$. Otherwise R_{i*} is equal to R_i.

Calculating the reflexive closure of a relation is trivial. Moreover, there exist efficient algorithms for calculating the transitive closure of a relation (Nuutila, 1995).

We can now turn to the completeness proof:

Lemma 14. *Let Θ be a saturated open branch of a tableau $\mathsf{Tab}(\phi)$ in the calculus of $\mathcal{H}(D)$ with restriction $(\mathcal{I})$. For any formula $\sigma\varphi \in \Theta$ such that $nom(\varphi) \subseteq nom(\phi)$, we have $\mathcal{M}^{\Theta}_{*}, i_{\Theta}(\sigma) \models \varphi$.*

Proof. The proof is by induction on the syntactic structure of φ.

- Cases $\varphi = p$, $\neg p$, $\psi \wedge \chi$, $\psi \vee \chi$, $a\psi$, $\mathsf{E}\psi$, $\mathsf{A}\psi$, $\mathsf{D}\psi$, and $\mathsf{B}\psi$ are handled just as in the hybrid S5 calculus, with the difference that references to nominal urfathers have to be replaced by references to inclusion urfathers.

- $\varphi = \Diamond_i\psi$. By Inclusion Urfather Closure (Lemma 13) and saturation of $(\Diamond)$, we have:
$$i_{\Theta}(\sigma)\Diamond_i\tau,\ \tau\psi\ \in \Theta$$
Then, by definition of R_i^{Θ} and induction hypothesis:
$$(i_{\Theta}(\sigma), i_{\Theta}(\tau)) \in R_i^{\Theta} \quad \text{and} \quad \mathcal{M}^{\Theta}_{*}, i_{\Theta}(\tau) \models \psi$$
Combining this, we obtain $\mathcal{M}^{\Theta}_{*}, i_{\Theta}(\sigma) \models \Diamond_i\psi$, as required.

- $\varphi = \Box_i\psi$. If there is no world σ_1 such that $(i_{\Theta}(\sigma), \sigma_1) \in R_{i*}^{\Theta}$ then this holds trivially. Otherwise, let σ_1 be such that $(i_{\Theta}(\sigma), \sigma_1) \in R_{i*}^{\Theta}$. We need to consider two subcases:
  - $(i_{\Theta}(\sigma), \sigma_1) \in R_i^{\Theta}$. By definition of R_i^{Θ} there must be a prefix τ_1 such that $\sigma_1 = i_{\Theta}(\tau_1)$ and $i_{\Theta}(\sigma)\Diamond_i\tau_1 \in \Theta$. Then by Inclusion Urfather Closure (Lemma 13), $i_{\Theta}(\sigma)\Box_i\psi \in \Theta$, and by closure under $(\Box)$, $\tau_1\psi \in \Theta$. Induction hypothesis entails $\mathcal{M}^{\Theta}, i_{\Theta}(\tau_1) \models \psi$, i.e., $\mathcal{M}^{\Theta}, \sigma_1 \models \psi$. From this it follows that $\mathcal{M}^{\Theta}, i_{\Theta}(\sigma) \models \Box_i\psi$.
  - $(i_{\Theta}(\sigma), \sigma_1) \in R_{i*}^{\Theta} \setminus R_i^{\Theta}$. If $i \in \mathcal{R}$ and $\sigma_1 = i_{\Theta}(\sigma)$, saturation by the rule (re) enforces the presence of ψ at the prefix $i_{\Theta}(\sigma)$, thus it follows that $\mathcal{M}^{\Theta}, i_{\Theta}(\sigma) \models \Box_i\psi$. If $i \in \mathcal{T}$, saturation by the rule (tr) gives us $\sigma_1\psi$.

□

Theorem 11. *The calculus of $\mathcal{H}(D)$ with restriction $(\mathcal{I})$ is complete.*

Proof. Let Θ be a saturated open branch of the tableau $\mathsf{Tab}(\varphi)$. Since $\sigma_0\varphi \in \Theta$, by Lemma 14 we get that φ is satisfiable. □

4.2.3 Discussion

Saturation of ($\Diamond$) and reusing existing accessibility statements Equivalence classes of prefixes and the loop-check ($\mathcal{I}$) enable us to define a stricter saturation condition for the rule ($\Diamond$) than in the calculus of Bolander and Blackburn. Indeed, in their calculus, ($\Diamond$) could be applied on $\sigma\Diamond\varphi$ and then on $\tau\Diamond\varphi$, even when $\sigma \sim \tau$. Here, we take into account the history of applications of ($\Diamond$) in the whole equivalence class of a given prefix to prevent such redundant diamond expansions. As a consequence, this requires that accessibility statements get copied to the representative of an equivalence class, which is done by ($\in$).

This has one unfortunate side-effect: our calculus must rely on the loop-check ($\mathcal{I}$) to terminate even for the language $\mathcal{H}(:)$, while Bolander and Blackburn's system doesn't. This is because copying accessibility statements invalidates the argument that prefixes make true smaller and smaller formulas as they are further away from the root prefix. Consider, for instance, the formula

$$n \;\wedge\; \Diamond\top \;\wedge\; \Box\Diamond\bot \;\wedge \Box n$$

(with $\top$ for $p \wedge \neg p$ for some $p \in \mathsf{PROP}$ and $\bot$ for $\neg\top$), where ($\Diamond$) is applied systematically before ($\in$). It does not terminate without the loop-check, but terminates in the system of Bolander and Blackburn.

Pattern-based blocking and model building The calculus of Kaminski and Smolka relies on pattern-based blocking, which is a restriction on diamond expansions that subsumes both the loop-check ($\mathcal{I}$) and the classwise saturation condition of ($\Diamond$). The idea of pattern-based blocking is to only expand a diamond formula if there is no previous diamond expansion in the branch where the created world makes true at least the same formulas. Adapted to our formalism, their definition of a *pattern* becomes:

$$P_\Theta(\sigma\Diamond_r\varphi) \equiv_{def} \{\Diamond_r\varphi\} \cup \{\Box_r\psi \;\mid\; \sigma\Box_r\psi \in \Theta\}$$

And the definition of saturation of the rule ($\Diamond$) becomes:

- ($\Diamond$) can not be applied to a premise $\sigma\Diamond_r\varphi$ on Θ if it has already been applied to $\tau\Diamond_r\varphi$ with
$$P_\Theta(\sigma\Diamond_r\varphi) \subseteq P_\Theta(\tau\Diamond_r\varphi)$$

With PBB, diamond formulas are individually blocked for a same prefix, while with IB, all diamond formulas of a given prefix are blocked at once. PBB blocks better than IB since the number of possible patterns in a tableau may be exponentially lower than the number of possible labellings of prefixes (Kaminski and Smolka, 2007).

The authors also present model building in a different way. In their approach, expressed in our terms, model building is done by adding missing links between nominal urfathers. In our approach, the model building step involves removing redundant nominal urfathers and repairing links. From an implementation point of

$$\frac{\sigma\Diamond_i^-\varphi}{\tau\Diamond_i\sigma, \tau\varphi}\,(\Diamond^-)^1 \qquad \frac{\sigma\Box_i^-\varphi, \tau\Diamond_i\sigma}{\tau\varphi}\,(\Box^-)$$

$$\frac{\sigma\Diamond\tau}{n_\Theta(\sigma)\Diamond n_\Theta(\tau)}\,(bridge) \qquad \frac{\sigma_1\Box_i^-\varphi, \sigma_0\Diamond_i\sigma_1}{\sigma_0\Box_i^-\varphi}\,(tr^-), i \in \mathcal{T}$$

$$\frac{\sigma\Box_i\varphi}{\sigma\Box_i^-\varphi}\,(sy), i \in \mathcal{S} \qquad \frac{\sigma\Box_i^-\varphi}{\sigma\Box_i\varphi}\,(sy^-), i \in \mathcal{S}$$

[1] The prefix τ is new to the branch.

Figure 4.8: Additional rules for $\mathcal{H}(\mathsf{D}, \Diamond^-)$ with symmetric relations.

view, the former approach makes more sense, even if it may lead to bigger models in very rare cases.

It is perfectly possible to obtain a decision procedure based on pattern-based blocking and the model building definition presented here, or conversely, on anywhere blocking and Kaminski and Smolka's model definition.

Loop-check The loop-check we use is also known as "subset blocking" and "anywhere blocking", notably in description logics tableau systems (Baader and Sattler, 2000). It is the same loop-check used in Bolander and Blackburn's calculus to handle the language $\mathcal{H}(:, \mathsf{E})$.

However, we restricted the definition of L^Θ, on which the definition of $\mathcal{I}$ relies, to formulas whose satisfaction definition depends on the world of evaluation, also excluding $\vee$-, $\wedge$-, D- and B-formulas, which, by saturation of the corresponding rules, are turned into smaller formulas pertaining to the same prefix. This makes inclusion blocking more efficient than in Bolander and Blackburn's calculus.

We have seen that ignoring the nominals introduced by the rule (B) is crucial to ensure termination. As these nominals only appear as positive literals, their presence does not interfere with the identification of a world with another one. For instance, consider the situation where we have two prefixes $\sigma < \tau$, with their sets of true formulas being $\{p, \Diamond q\}$ and $\{\Diamond q, n\}$ respectively, and n being a new nominal introduced by (B). Here, τ is blocked by σ. It is safe to block like this because it is guaranteed that $\neg n$ never occurs on the branch.

Although anywhere blocking may be advantageously traded for pattern-based blocking, the definition of L^Θ is going to be useful for definition the loop-check used in the next calculus.

4.3 Adding symmetric and converse modalities

We now extend the calculus of $\mathcal{H}(\mathsf{D})$ seen in the previous section so that it handles converse modalities $\Diamond_i^-$ and symmetric modalities. The additional rules are given in Figure 4.8. We call the resulting calculus *the calculus of* $\mathcal{H}(D,\Diamond^-)$.

The rules $(\Diamond^-)$ and $(\Box^-)$ should be clear since they behave like $(\Diamond)$ and $(\Box)$. However notice that accessibility relations introduced are always forward. The saturation conditions for these rules is also similar to the ones of their non-converse counterparts. Rule (tr^-) is required to enable transitivity on converse modalities, and rules (sy) and (sy^-) enforce symmetry.

The rule $(bridge)$ however may need to be explained a little more in detail. Indeed, it is now necessary to somehow *normalise* accessibility statements so that they only involve nominal urfathers. Let us use the phrase *forward constraints* to refer to formulas of the shape $\Box\varphi$, *backwards constraints* for formulas of the shape $\Box^-\varphi$, and *box constraints* for both shapes.

In the calculus of $\mathcal{H}(\mathsf{D})$, $(bridge)$ is not needed because no backwards constraint occurs. Forward constraints are propagated along accessibility statements and then copied to nominal urfathers thanks to $(\in)$, as needed. In the calculus of $\mathcal{H}(\mathsf{D},\Diamond^-)$ without the $(bridge)$ rule, a backwards constraint may remain unpropagated. Indeed, by looking at the premises of $(\neg\Diamond^-)$, one can see that nothing will happen if an accessibility statement arrives in the prefix. For instance, the following formula does not yield a closed tableau without the $(bridge)$ rule:

$$\neg p \;\wedge\; \Diamond n \;\wedge\; n : \Box^- p$$

However we need to change a few extra things in the calculus. The following example shows that the loop-check $(\mathcal{I})$ pose a problem with converse modalities.

Consider the unsatisfiable formula $p \wedge \mathsf{A}\Diamond(p \wedge \Box^-\Box^-\neg p)$. Under restriction $(\mathcal{I})$, a saturated tableau with this formula as root does not close because the first prefix generated by the rule $(\Diamond)$ is blocked by the root prefix:

$$\begin{array}{l|l}
\sigma_0 p \wedge \mathsf{A}\Diamond(p \wedge \Box^-\Box^-\neg p) & \text{initial node} \\
\sigma_0 p & (\wedge) \\
\sigma_0 \mathsf{A}\Diamond(p \wedge \Box^-\Box^-\neg p) & \\
\sigma_0 \Diamond(p \wedge \Box^-\Box^-\neg p) & (\mathsf{A}) \\
\sigma_0 \Diamond \sigma_1 & (\Diamond) \\
\sigma_1 p \wedge \Box^-\Box^-\neg p & \\
\sigma_1 p & (\wedge) \\
\sigma_1 \Box^-\Box^-\neg p & \\
\sigma_0 \Box^-\neg p & (\Box^-) \\
\sigma_1 \Diamond(p \wedge \Box^-\Box^-\neg p) & (\mathsf{A})
\end{array}$$

Without $(\mathcal{I})$, we could actually close the tableau by continuing the branch:

$$\begin{array}{l|l}
\sigma_1 \Diamond \sigma_2 & (\Diamond) \\
\sigma_2 p \wedge \Box^- \Box^- \neg p & \\
\sigma_2 p & (\wedge) \\
\sigma_2 \Box^- \Box^- \neg p & \\
\sigma_1 \Box^- \neg p & (\Box^-) \\
\sigma_0 \neg p & (\Box^-) \\
\otimes &
\end{array}$$

In other words, restriction ($\mathcal{I}$) is too strict in the presence of converse modalities. We have to define a new restriction that ensures completeness without sacrificing termination. The loop-check we will use is based on repeating chains of prefixes generated by the rules ($\Diamond$) and ($\Diamond^-$), as in (Bolander and Blackburn, 2007a) and (Kaminski and Smolka, 2009a). To be able to refer to chain of prefixes, we need the following notation:

Definition 28. *If a prefix τ has been introduced in a branch Θ by applying one of the rules ($\Diamond$) and ($\Diamond^-$) to a premise $\sigma\varphi$ then we write $\sigma \triangleright_\Theta \tau$. We use $\triangleright^*_\Theta$ to denote the transitive and reflexive closure of the relation $\triangleright_\Theta$.*

We now prepare the definition of the new loop-check:

Definition 29. *If σ and τ are two prefixes in a branch Θ such that $L^\Theta(\sigma) = L^\Theta(\tau)$ and not $\sigma \sim_\Theta \tau$, we call them* twins *on Θ.*

Definition 30. *A prefix σ in Θ is said to be* unblocked *if there is no pair of twins τ and τ' such that $\tau \triangleright^*_\Theta \tau' \triangleright^*_\Theta \sigma$.*

Note that if σ is unblocked on Θ and $\sigma' \triangleright^*_\Theta \sigma$ then σ' is necessarily also unblocked. The loop-check is defined as follows:

Definition 31. *(**Loop-check** ($\mathcal{C}$)) The rules ($\Diamond$) and ($\Diamond^-$) are only applied to a formula $\sigma\varphi$ on a branch if σ is unblocked on that branch.*

We named this loop-check ($\mathcal{C}$) as in "chain" since this restriction relies on information present in the ancestry chain of a given prefix.

4.3.1 Termination

At this point, we actually need few steps to show termination of the calculus of $\mathcal{H}(\mathsf{D}, \Diamond^-)$. Saturation by rules ($\Diamond$) and ($\Diamond^-$) implies the the following result on the relation $\triangleright_\Theta$:

Lemma 15. *The graph $(P^\Theta, \triangleright_\Theta)$, where P^Θ is the set of prefixes linked by the relation $\triangleright_\Theta$, is a forest of finitely branching trees.*

We now can prove termination of the calculus of $\mathcal{H}(\mathsf{D}, \Diamond^-)$ with restriction ($\mathcal{C}$):

Theorem 12. *Any tableau in the calculus of $\mathcal{H}(D, \Diamond^-)$ constructed under restriction ($\mathcal{C}$) is finite.*

Proof. Suppose there is an infinite tableau. By following the same argument as the one of the proof of Theorem 10, we know that there are infinitely many prefixes in the branch, and at the same time, there can be only finitely many applications of the rules (D) and (E). This implies that there are infinitely many applications of ($\Diamond$) and ($\Diamond^-$).

Given Lemma 15 and König's lemma, there is one infinite chain of prefixes generated by ($\Diamond$) or ($\Diamond^-$):

$$\sigma_n \triangleright_\Theta \sigma_{n+1} \triangleright_\Theta \sigma_{n+2} \triangleright_\Theta \cdots .$$

Now, there is a maximal number of applications of ($\Diamond$) and ($\Diamond^-$) in a given equivalence class, by definition of the saturation of these rules and the quasi-subformula property. Let us call this number d. Moreover, we know from Lemma 8 that there can only be finitely many different sets $L^\Theta(\sigma)$ for σ on the branch Θ. Let m be this number.

Let us consider the prefix $\sigma_{n+d(m+1)+1}$ of the previous chain. It has been introduced by ($\Diamond$) or ($\Diamond^-$) applied on prefix of rank $n + d(m + 1)$ on Θ. Because of restriction ($\mathcal{C}$), $\sigma_{n+d(m+1)}$ must then be unblocked on Θ'. However, there exist two prefixes σ_l and σ_k with $l, k < n + d(m + 1)$, such that $L^\Theta(\sigma_l) = L^\Theta(\sigma_k)$ without $\sigma_l \sim_\Theta \sigma_k$, that is to say σ_l and σ_k are twins. This contradicts $\sigma_{n+d(m+1)}$ being unblocked on Θ', which makes the existence of such an infinite chain impossible. □

4.3.2 Completeness

In the previous calculus, inclusion urfathers were used to block other prefixes, and also as elements of the model built from a saturated open branch. But we have seen that a loop-check based on inclusion urfathers is not adequate for completeness in the case of the calculus of $\mathcal{H}(\mathsf{D}, \Diamond^-)$, so we now rely on a weaker loop-check based on unblocked prefixes. Nonetheless, unblocked prefixes cannot be used as elements of an extracted model, since two unblocked prefixes can make true the same nominal. This situation is in fact very common, as in the following example with the formula a (with $a \in \mathsf{NOM}$):

$$\begin{array}{r|l} \sigma_0 a & \text{initial branch} \\ \sigma_1 a & \end{array}$$

σ_0 and σ_1 are both unblocked prefixes, thus unblocked prefixes can not be chosen, as were inclusion urfathers, as elements of a model extracted from a saturated open branch. Therefore, we introduce another kind of urfather, the unblocked urfather:

Definition 32. *Let σ be a prefix occurring in a branch Θ. The* unblocked urfather *of σ on Θ, written $u_\Theta(\sigma)$, is the smallest prefix τ satisfying:*

1. $L^\Theta(\sigma) = L^\Theta(\tau)$
2. *τ is unblocked.*

Such a prefix does not necessarily exist, thus u_Θ is only a partially defined mapping. A prefix σ is called an unblocked urfather *in Θ if $\sigma = u_\Theta(\tau)$ for some prefix τ.*

We write $\sigma \in \mathrm{dom}(u_\Theta)$ when $u_\Theta(\sigma)$ is defined.

In other words, the unblocked urfather of a prefix is its smallest unblocked twin. Note that there is no guarantee that it exists, and if it exists, no guarantee that it is on the same chain of ancestry. Thus, a prefix can be blocked without being represented in the possible model.

The following result ensures that successors of unblocked urfathers can be represented in the model by an (other) unblocked urfather:

Lemma 16. *Let σ be unblocked on a branch Θ. If $\sigma \triangleright_\Theta \tau$ then $\tau \in \mathrm{dom}(u_\Theta)$.*

Proof. Assume $\sigma \triangleright_\Theta \tau$ where σ is unblocked on Θ. If τ is unblocked on Θ then $\tau \in \mathrm{dom}(u_\Theta)$. So assume that τ is not unblocked. Then there must exist a pair of distinct twins γ, γ' with $\gamma \triangleright^*_\Theta \gamma' \triangleright^*_\Theta \tau$. Since σ is unblocked we can not have both $\gamma \triangleright^*_\Theta \sigma$ and $\gamma' \triangleright^*_\Theta \sigma$. Since $\sigma \triangleright_\Theta \tau$ this implies $\gamma' = \tau$. Thus τ has γ as a twin, and since necessarily $\gamma \triangleright^*_\Theta \sigma$ we get that γ is unblocked. Since γ is a candidate to being the unblocked urfather of τ, $u_\Theta(\tau)$ is defined. □

As in the previous calculus, nominals introduced by ($\neg$D) are not taken into account when it comes to defining unblocked urfathers. We now prove, as before, the properties Urfather Closure, Urfather Equality and Urfather Characterisation.

Lemma 17 (**Unblocked Urfather Closure**)**.** *Let σ be a prefix in a saturated branch Θ where ($\in$) is applied, with $\sigma \in \mathrm{dom}(u_\Theta)$ and φ a model-relevant local formula. If $\sigma\varphi \in \Theta$, then $u_\Theta(\sigma)\varphi$.*

Proof. Proof similar to the one of Lemma 13. □

Lemma 18 (**Unblocked Urfather Equality**)**.** *Let Θ be a saturated branch. If $\sigma, \tau \in \mathrm{dom}(u_\Theta)$ and $\sigma \sim_\Theta \tau$, then $u_\Theta(\sigma) = u_\Theta(\tau)$.*

Proof. By Lemma 4, since $\sigma \sim_\Theta \tau$, then $L^\Theta(\sigma) = L^\Theta(\tau)$, thus $u_\Theta(\sigma) = u_\Theta(\tau)$. □

Lemma 19 (**Unblocked Urfather Characterisation**)**.** *Let Θ be a branch. Then σ is an unblocked urfather if and only if $u_\Theta(\sigma) = \sigma$.*

Proof. For the "only if" direction, suppose σ is an unblocked urfather, i.e., there exists a prefix τ such that $\sigma = u_\Theta(\tau)$. That is, σ is the smallest unblocked prefix such that $L^\Theta(\sigma) = L^\Theta(\tau)$. The prefix $u_\Theta(\sigma)$ has the property of being the smallest unblocked prefix such that $L^\Theta(u_\Theta(\sigma)) = L^\Theta(\sigma)$. As a consequence, $u_\Theta(\sigma)$ is also the unblocked urfather of τ, which means $u_\Theta(\sigma) = \sigma$. □

We have all the tools to describe how to build a model from a saturated open branch in the present calculus:

Definition 33. *Given an open, saturated branch Θ with root $\sigma_0\varphi_0$ in the calculus of $\mathcal{H}(D,\Diamond^-)$, we define a model $\mathcal{M}^\Theta = (W^\Theta, (R_i^\Theta)_{i<n}, V^\Theta)$ where:*

$$\begin{aligned} W^\Theta &= \{\sigma \mid \sigma \textit{ is an unblocked urfather on } \Theta\} \\ R_i^\Theta &= \{(u_\Theta(\sigma), u_\Theta(\tau)) \mid \sigma\Diamond_i\tau \textit{ occurs on } \Theta \textit{ and } \sigma,\tau \in \mathrm{dom}(u_\Theta)\} \\ V^\Theta(s) &= \{u_\Theta(\sigma) \mid \sigma s \textit{ occurs on } \Theta \textit{ and } \sigma \in \mathrm{dom}(u_\Theta)\}. \end{aligned}$$

That $V^\Theta(a)$ is a singleton for any nominal a follows from Urfather Equality (Lemma 18). As before, $\mathcal{M}_*^\Theta$ is the model in which all relations R_i of $\mathcal{M}^\Theta$ are closed respectively for reflexivity, transitivity and symmetry when $i \in \mathcal{R}$, $i \in \mathcal{T}$ and $i \in \mathcal{S}$. We write R_{i*} for the appropriate closure of R_i.

Notice how we define W^Θ: this is because not all prefixes of the branch have an unblocked urfather. We can now prove completeness for the calculus of $\mathcal{H}(\mathsf{D},\Diamond^-)$ with restriction $(\mathcal{C})$.

Lemma 20. *Let Θ be a saturated open branch of a tableau $\mathsf{Tab}(\phi)$ in the calculus of $\mathcal{H}(D,\Diamond^-)$ with restriction $(\mathcal{C})$. For any formula $\sigma\varphi \in \Theta$ such that $\sigma \in \mathrm{dom}(u_\Theta)$ and $nom(\varphi) \subseteq nom(\phi)$, we have $\mathcal{M}_*^\Theta, u_\Theta(\sigma) \models \varphi$.*

Proof. Again, the proof is by induction on the syntactic structure of φ:

- φ has one of the forms p, $\neg p$, $\psi \wedge \chi$, $\psi \vee \chi$, $\mathsf{A}\psi$, $\mathsf{B}\psi$ or $\Box_i\psi$ with i possibly in $\mathcal{R}$ or $\mathcal{T}$. We can directly reuse the previously given proof by simply replacing references to i_Θ by u_Θ and references to "inclusion urfather" by "unblocked urfather". This is because we still have the Urfather Characterisation property (Lemma 19).

- $\varphi = \sigma\Box_i\psi \in \Theta$ with $i \in \mathcal{S}$. If $(\sigma_1, u_\Theta(\sigma)) \in R_{i*}^\Theta$, then by saturation by (sy), $(bridge)$ and $(\Box^-)$, $\sigma_1\psi \in \Theta$.

- φ is of the form $\Diamond_i\psi$, $\mathsf{E}\psi$ or $\mathsf{D}\psi$. We can also reuse the previous proof, adding that when a prefix generating rule is applied to the premise $u_\Theta(\sigma)\varphi$ to produce a conclusion $\tau\chi$, then $\tau \in \mathrm{dom}(u_\Theta)$ (Lemma 16), which enables us to use the induction hypothesis.

- $\varphi = \Diamond_i^-\psi$. Similar to $\varphi = \Diamond_i\psi$, with adjustments described in the previous case.

- $\varphi = \Box_i^-\psi$. If there is no σ_1 such that $(\sigma_1, u_\Theta(\sigma)) \in R_i^\Theta$ then the property holds. Otherwise, let such σ_1 be chosen arbitrarily. We need to prove that $\mathcal{M}^\Theta, \sigma_1 \models \psi$. By definition of R_i^Θ, there exist prefixes τ, τ_1 such that $\sigma_1 = u_\Theta(\tau_1)$, $u_\Theta(\sigma) = u_\Theta(\tau)$ and $\tau_1\Diamond_i\tau \in \Theta$. By saturation of the $(bridge)$ rule, we have $n_\Theta(\tau_1)\Diamond_i n_\Theta(\tau)$ (1). By definition of Unblocked Urfathers, in particular the fact that the unblocked urfather of a prefix is a *twin* of it, we have that $u_\Theta(\sigma)$ and $u_\Theta(\tau)$ are twins, therefore $n_\Theta(\tau)$ also, thus $n_\Theta(\tau)\Box_i^-\psi$ (2). By saturation of $(\Box^-)$ on (2) and (1), we get $n_\Theta(\tau_1)\psi$. Then, by induction hypothesis we have $\mathcal{M}^\Theta, u_\Theta(n_\Theta(\tau)) \models \psi$, which, is in fact $\mathcal{M}^\Theta, \sigma_1 \models \psi$ as needed.

- $\varphi = \Box_i^- \psi$ when $i \in \mathcal{R} \cup \mathcal{T} \cup \mathcal{S}$ is handled as previously, involving the rules (*bridge*) and (tr^-) and (sy^-) when needed.

□

With a similar argument to the one of Theorem 11, we can claim:

Theorem 13. *The calculus of* $\mathcal{H}(\mathsf{D}, \Diamond^-)$ *with restriction* ($\mathcal{C}$) *is complete.*

4.3.3 Discussion

Normalising accessibility statements For this calculus, the rule (*bridge*) is crucial for normalizing accessibility statements. This rule is not requires in the calculus of Bolander and Blackburn because of the way formulas are copied to every element of equivalence classes. In that setting, backwards constraints would first be copied to the adequate prefix before being propagated. Thus (*bridge*) enables us to keep a small footprint of copied formulas.

Loop-check As in the calculus of Bolander and Blackburn, and the one of Kaminski and Smolka, anywhere or pattern-based blocking cannot be used because they interact badly with converse modalities. We thus rely on a loop-check that uses only information present in the "ancestry" of a given prefix. Of course, this is far from satisfactory since this yields a much bigger lower bound on the maximal size of the tableau. Tableaux systems for description logics with inverse roles also rely on blocking conditions that only use information of the ancestry of a given node, and this in spite of intensive research in the field (see, for instance, (Horrocks and Sattler, 1999) and (Tsarkov et al., 2007)).

Symmetric relations Symmetric relations cannot be handled in the calculus of $\mathcal{H}(\mathsf{D})$ shown in Section 4.2, where the loop-check ($\mathcal{I}$) works by subset checking. The reason lies in the relations that we can build from an open branch, in the presence of this loop-check:

$$R_i^\Theta = \{(\sigma, i_\Theta(\tau)) \mid \sigma \in W^\Theta \text{ and } \sigma \Diamond_i \tau \text{ occurs on } \Theta\}$$

As $i_\Theta(\tau)$ can make true more formulas than τ, it can have more box constraints, thus requiring more information to be present at σ, which is not guaranteed. Inclusion blocking worked well when only forward constraints are present, but here completeness is clearly broken. We find again that blocking and model building using twins is essential for a calculus with converse or with symmetric relations.

4.4 Non-terminating tableaux

Let us now consider extensions to the last calculus for which we will not prove termination. In particular, let us start from the calculus of $\mathcal{H}(\mathsf{D}, \Diamond^-)$ without any loop-check. Moreover, when considering completeness of these extensions, we will no longer build a model from an open branch by Definitions 27 or 33, but we will use nominal urfathers as worlds of the model and members of the accessibility relations. This is to avoid the addition of spurious links, enabling us to guarantee completeness of these extensions.

The three extensions we are going to see are role inclusion, functional and injective modalities, and the down-arrow binder.

4.4.1 Role inclusion

Role inclusion is the ability to say that an accessibility relation is included in another one. This specification often appears in description logics, as part of the TBox. In the case of hybrid logics, let us internalize this constraint and add the following formula kind to any of the last two sublanguages seen in this chapter:

$$r \sqsubseteq s$$

with $r, s \in \mathsf{REL}$.

The semantics of $r \sqsubseteq s$ are given by:

$$\mathcal{M}, w \models r \sqsubseteq s \quad \text{iff} \quad R_r \subseteq R_s$$

This new syntax enables to write the following shortcuts:

$$\begin{array}{lcl} r \sqsubseteq \bigsqcup_{1 \leq i \leq n} s_i & \equiv & (r \sqsubseteq s_1) \vee \ldots \vee (r \sqsubseteq s_n) \\ (\bigsqcup_{1 \leq i \leq n} s_i) \sqsubseteq r & \equiv & (s_1 \sqsubseteq r) \wedge \ldots \vee (s_n \sqsubseteq r) \\ (r = \bigsqcup_{1 \leq i \leq n} s_i) & \equiv & (r \sqsubseteq \bigsqcup_{1 \leq i \leq n} s_i) \wedge (\bigsqcup_{1 \leq i \leq n} s_i) \sqsubseteq r) \end{array}$$

This last syntax enables to define a relation as being exactly the union of several others.

The tableau rule associated to role inclusion is:

$$\boxed{\frac{\sigma \Diamond_r \tau \ , \ \gamma(r \sqsubseteq s)}{\sigma \Diamond_s \tau} \ (\sqsubseteq)}$$

It should be clear that for any saturated open branch Θ where $(\sqsubseteq)$ has been applied, the extracted model $\mathcal{M}$, as defined in the introduction of this section, will comply with the condition $\mathcal{M}^r \subseteq \mathcal{M}^s$, for all formulas $\sigma(r \sqsubseteq s) \in \Theta$.

Figure 4.9 shows an example mixing role inclusion and transitivity, yielding a closed tableau.

$\sigma_0((r \sqsubseteq s) \wedge (\Diamond_r\Diamond_r p) \wedge (\Box_s \neg p))$	initial node
$\sigma_0(r \sqsubseteq s)$	$(\wedge)$
$\sigma_0 \Diamond_r \Diamond_r p$	
$\sigma_0 \Box_s \neg p$	
$\sigma_0 \Diamond_r \sigma_1$	$(\Diamond)$
$\sigma_1 \Diamond_r p$	
$\sigma_1 \Diamond_r \sigma_2$	$(\Diamond)$
$\sigma_2 p$	
$\sigma_0 \Diamond_s \sigma_1$	$(\sqsubseteq)$
$\sigma_1 \Diamond_s \sigma_2$	$(\sqsubseteq)$
$\sigma_1 \neg p$	$(\Box)$
$\sigma_1 \Box_s \neg p$	(tr)
$\sigma_2 \neg p$	$(\Box)$
$\otimes$	

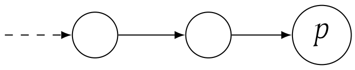
Figure 4.9: Closed tableau for $(r \sqsubseteq s) \wedge (\Diamond_r\Diamond_r p) \wedge (\Box_s \neg p), \quad s \in \mathcal{T}$

4.4.2 Functional and injective modalities

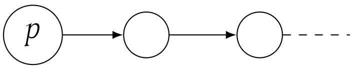
Let us expand the signature given in introduction of this chapter by adding $\mathcal{F}$ and $\mathcal{F}^-$, two subsets of REL that respectively correspond to modalities whose accessibility relation is interpreted as functional and injective.

A modality i is *functional* (resp. *injective*) if, for a given world w in a model M, there exists at most one world v such that $(w, v) \in R_i$ (resp. $(v, w) \in R_i$). In the literature, functional and injective relations are also called *right-unique* and *left-unique*.

Contrarily to fragments previously seen in this chapter, modal logic with converse modalities and functional modalities no longer has the finite model property. For instance, the formula $p \wedge \mathsf{A}\Diamond_i^- \neg p$, with i being interpreted as a functional accessibility relation, is satisfiable only in an infinite model:



A similar case can be built without converse modalities but with an injective one, with the formula $p \wedge \mathsf{A}\Diamond^i \neg p$:



In Description Logics, the presence of functional modalities is indicated by the letter $\mathcal{F}$, and are referred to by the expression “functional restrictions” (De Giacomo and Lenzerini, 1994; Horrocks et al., 1999b).

Injective modalities can be useful to enforce a tree-like structure on the model. Indeed, a model of a modal formula without nominals, nor universal or difference modalities, whose only accessibility relation is an injective relation, has the shape of a tree. Bolander and Blackburn (2007b) indeed call injective modalities “tree-like”. These tree structures are useful for instance to represent XML trees. On the other hand, functional modalities are appropriate to represent deterministic automatons.

To handle these modalities, we add the following rules to our system:

$$\frac{\sigma_0 \Diamond_i \sigma_1, \sigma_0 \Diamond_i \sigma_2}{\sigma_1 n, \sigma_2 n}\ (f), i \in \mathcal{F} \qquad \frac{\sigma_1 \Diamond_i \sigma_0, \sigma_2 \Diamond_i \sigma_0}{\sigma_1 n, \sigma_2 n}\ (f^-), i \in \mathcal{F}^-$$

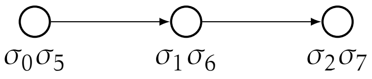
For each rule, n is a new nominal, and the saturation condition is that $\sigma_1 \not\sim \sigma_2$.

Both rules should be read as: whenever there are two prefixes accessible by $\Diamond_i$ from the same prefix, that do not belong to the same equivalence class, we “fix” that with new nominals.

Using nominals to enforce these frame properties is convenient in the context of hybrid logics, as these new nominals “automatically” repair functionality or injectivity. Example of Figure 4.10 shows this happening: we initially have two strand of prefixes linked by a functional accessibility relation. As soon as both origin prefixes σ_0 and σ_5 receive the nominal a, the rest of the prefixes have to be “merged” accordingly.

$\dots$	
$\sigma_0 \Diamond \sigma_1$	
$\sigma_1 \Diamond \sigma_2$	
$\sigma_5 \Diamond \sigma_6$	
$\sigma_6 \Diamond \sigma_7$	
$\sigma_0 a$	
$\sigma_5 a$	
$\sigma_0 \Diamond \sigma_6$	$(\in)$
$\sigma_1 n_0$	(f)
$\sigma_6 n_0$	
$\sigma_1 \Diamond \sigma_7$	$(\in)$
$\sigma_2 n_1$	(f)
$\sigma_7 n_1$	

(a)



(b)

Figure 4.10: (a) End of a tableau branch, where $\Diamond$ is interpreted as a functional relation. (b) model

4.4.3 The down-arrow binder

Let us finish by adding the down-arrow binder to the language $\mathcal{H}(\mathsf{D}, \Diamond^-)$, so as to get $\mathcal{H}(\mathsf{D}, \Diamond^-, \downarrow)$. We recall the semantics of the down-arrow binder:

$$\langle W, R, V \rangle, w \models {\downarrow} a.\varphi \quad \text{iff} \quad \langle W, R, V_a^w \rangle, w \models \varphi$$

That is, the formula $\downarrow a.\varphi(a)$ binds the name a to the current evaluation world, and then asserts $\varphi(a)$. We know that the language obtained is undecidable, so of course no terminating tableaux system can be designed for it.

The binder can easily be handled as an extra tableau rule:

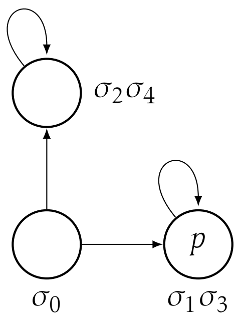
$$\boxed{\frac{\sigma{\downarrow}x.\varphi}{\sigma n, \sigma\varphi[x \leftarrow n]}\ (\downarrow)}$$

In the rule above, n is a new nominal and $\varphi[x \leftarrow n]$ is the formula obtained by replacing all occurrences of x by n. So we do not "rebind" the nominal x, but we introduce a new nominal and formula to mimic this binding. Soundness and completeness of the calculus of $\mathcal{H}(\mathsf{D}, \Diamond^{-})$ without loop-check and extended with $(\downarrow)$ is obvious.

Moreover, let us modify the definition of $\mathsf{Tab}(\varphi)$, so that bound nominals do not appear in the root of the tableau. Let us see the calculus in action with the example of Figure 4.11.

$\sigma_0(\Diamond p \ \wedge\ \Diamond\neg p \ \wedge\ \Box{\downarrow}x.\Diamond x)$	initial node
$\sigma_0\Diamond p$	$(\wedge)$
$\sigma_0\Diamond\neg p$	
$\sigma_0\Box{\downarrow}x.\Diamond x$	
$\sigma_0\Diamond\sigma_1$	$(\Diamond)$
$\sigma_1 p$	
$\sigma_0\Diamond\sigma_2$	$(\Diamond)$
$\sigma_2\neg p$	
$\sigma_1{\downarrow}x.\Diamond x$	$(\Box)$
$\sigma_2{\downarrow}x.\Diamond x$	$(\Box)$
$\sigma_1 n_1$	$(\downarrow)$
$\sigma_1\Diamond n_1$	
$\sigma_1\Diamond\sigma_3$	$(\Diamond)$
$\sigma_3 n_1$	
$\sigma_2 n_2$	$(\downarrow)$
$\sigma_2\Diamond n_2$	
$\sigma_2\Diamond\sigma_4$	$(\Diamond)$
$\sigma_4 n_2$	

(a)



(b)

Figure 4.11: (a) Tableau for $(\Diamond p \ \wedge\ \Diamond\neg p \ \wedge\ \Box{\downarrow}x.\Diamond x)$ (b) model

Chapter 5

Logics with counting

In this chapter we explore the addition of *counting operators* to a logic. That is to say, explicit operators to count the number of elements that a model might include in the extension of a formula.

As we will see, the idea of forcing the number of elements of the model making true a given formula has been studied in the following contexts:

- first-order logic with equality and first-order with generalized quantifiers
- graded modal logic
- description logic

We will, in general, discuss decidability and complexity results of the logics with counting we will encounter.

Then, we will introduce a simple instance of a logic with counting called $\mathcal{MLC}$ (Modal Logic with Counting), that is, the basic modal logic augmented with a counting operator $(\varphi \geq n)$. As we will explain, this language is related to hybrid logics and the universal modality. We investigate the expressive power of this logic via bisimulations, and show that it is incomparable with graded modal logic.

Finally, we will define a new reasoning task that retrieves the cardinality bound of the extension of a given input formula, and provide an algorithm to solve it. This procedure works for any logic with counting for which a decision and model building procedure is available.

5.1 Counting operators

We call *counting operator* operators that can express "at least" or "at most" constraints. If a logic has both constraints, then it can also of course express "exactly" constraints. These kind of operators have been called *generalized quantifiers* and were originally introduced by Mostowski (1957). Included in this family of generalized quantifiers are other ones, like "most x are y", but we will not discuss them here. Such constraints are studied in modal logics for instance in (Pacuit and Salame, 2004).

5.1.1 First-order logics can count (with help)

Even without equality, first-order logic (FOL) can express “at least” constraints. It suffices to rely on encoding of numbers with predicates to ensure that different variables are interpreted by different elements in a model. If we want to represent “at least four apples”, for instance, we could write:

$$
\begin{array}{l}
\exists x_0, x_1, x_2, x_3. \\
\qquad\qquad \begin{array}{crcr}
(& P_0(x_0) & \wedge & P_1(x_0) \\
\wedge & \neg P_0(x_1) & \wedge & P_1(x_1) \\
\wedge & P_0(x_2) & \wedge & \neg P_1(x_2) \\
\wedge & \neg P_0(x_3) & \wedge & \neg P_1(x_3) \\
\wedge & Apple(x_0) & & \\
\wedge & Apple(x_1) & & \\
\wedge & Apple(x_2) & & \\
\wedge & Apple(x_3) & &)
\end{array}
\end{array}
$$

However, “at most” constraints cannot be expressed in FOL without equality:

Theorem 14. *FOL without equality can not express “at most” constraints.*

Proof. It is always possible to turn a model into a bigger one by adding “clones” of individuals. Indeed, let φ be a FOL formula satisfied in a model $\langle D, I\rangle$, i.e. $\langle D, I\rangle \models \varphi$, and let i be an individual of $\langle D, I\rangle$. Let $\langle D, I\rangle^{+i}$ be the model obtained by adding the individual i', that agrees with i on all predicates. Since we get again that $\langle D, I\rangle^{+i} \models \varphi$ (this can be verified by going through the definition of $\models$), there is no FOL formula that can express an “at most” constraint. □

On the other hand, FOL with equality can express any finite counting quantifier. If we want to express “at least four apples”, we can write:

$$\exists x_0, x_1, x_2, x_3.((\bigwedge_{i<j} x_i \neq x_j) \wedge (\bigwedge_{i} Apple(x_i)))$$

And we can also express “at most four apples” with the formula:

$$\forall x_0, x_1, x_2, x_3, x_4.((\bigwedge_{i} Apple(x_i)) \rightarrow (\bigvee_{i<j} x_i = x_j))$$

Thus for both constraints of the kind “at most n” and “at least n”, a formula needs to be of size quadratic in function of n.

We can also add first-class counting quantifiers to FOL. This was done already by Mostowski (1957), where he introduced *generalized quantifiers*, enabling to express statements such as “at least”, “at most”, “more x than y”, etc. We are particularly interested in the following two quantifiers (with $n \in \mathbb{N}$):

$$\begin{array}{lll} \langle D,I\rangle \models \exists^{\geq n}x.\varphi & \text{iff} & |\{d \mid \langle D,I_x^d\rangle \models \varphi\}| \geq n \\ \langle D,I\rangle \models \exists^{\leq n}x.\varphi & \text{iff} & |\{d \mid \langle D,I_x^d\rangle \models \varphi\}| \leq n \end{array}$$

We have that $\exists^{\geq 1}x.\varphi$ is equivalent to $\exists x.\varphi$ and $\exists^{\leq 0}x.\neg\varphi$ is equivalent to $\forall x.\varphi$. We can now express more succinctly our requirement of having at least 4 apples:

$$\exists^{\geq 4}x.Apple(x).$$

Of course these two counting quantifiers can in their turn define other ones:

$$\begin{array}{lll} \exists^{>n}\varphi & \equiv & \exists^{\geq n+1}\varphi \\ \exists^{<n}\varphi & \equiv & \exists^{\leq n-1}\varphi \\ \exists^{=n}\varphi & \equiv & (\exists^{\geq n}\varphi) \wedge (\exists^{\leq n}\varphi) \\ \exists^{\neq n}\varphi & \equiv & \neg(\exists^{=n}\varphi) \end{array}$$

Note that this logic does not need equality anymore to express counting.

Now, when it comes to computational inference, we are mostly interested in decidable languages, and, if possible, well-behaved ones. Full FOL with counting quantifiers is clearly undecidable.

However, some decidable fragments of first-order logic enriched with counting quantifiers have been studied. The two-variable fragment with counting has been shown to be decidable by Grädel et al. (1997a) and independently by Pacholski et al. (1997, 2000). Pratt-Hartmann (2005, 2010) showed that the satisfiability problem of this fragment is in *NEXPTIME*. Since the same fragment without counting is known to be *NEXPTIME*-hard (Grädel et al., 1997b), we get *NEXPTIME*-completeness for its counting counterpart. The complexity of the one-variable fragment is also studied by Pratt-Hartmann (2008): its satisfiability problem is *NP*-complete.

5.1.2 Graded modal logic

The idea of counting in modal logics also goes a long way back. Fine (1972) introduced the notion of *graded modalities*. The semantic definition of the graded modality $M_n\varphi$ is given by the condition

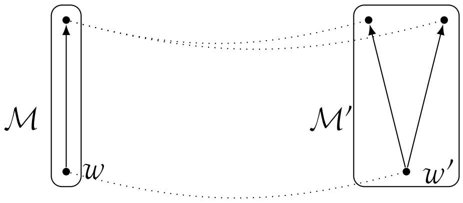
$$\mathcal{M}, w \models M_n\varphi \iff |\{w' \mid R(w,w') \text{ and } \mathcal{M}, w' \models \varphi\}| \geq n.$$

That is, $M_n\varphi$ holds at world w when there are at least n successors of w at which φ is true. Notice that this is a different kind of counting from the one we saw with first-order logic. In the present case, we are dealing with "successor counting". This fits very well into the mindset of modal logics with relational semantics, that is, talking about graphs from an internal perspective.

In more recent literature, graded modalities are written differently. Their usual syntax is $\langle r\rangle_{\geq n}\varphi$ and $\langle r\rangle_{\leq n}\varphi$. An alternative notation, used for instance by Kaminski et al. (2009b); Kaminski and Smolka (2010c); Pacuit and Salame (2004), is $\langle r\rangle_n$ for "at least $n+1$ r-successors make φ true" and $[r]_n\varphi \equiv$ for "every r-successor except at most n make φ true".

With this last notation, we have that $\Diamond_0\varphi \equiv \Diamond\varphi$ and $\Box_0\varphi \equiv \Box\varphi$ and the following duality holds: $\neg\Diamond_n\varphi \equiv \Box_n\neg\varphi$. Clearly, the two alternative notations are equivalent, as $\langle r\rangle_n\varphi \equiv \langle r\rangle_{\geq n+1}\varphi$ and $[r]_n\varphi \equiv \langle r\rangle_n\neg\varphi$. From now on we will use the former notation.

Graded modal logic is of course more expressive than modal logic (De Rijke, 2000). The following two models are bisimilar (Definition 14), as represented by the dotted lines, thus they make true the same sets of modal formulas (Theorem 5):



However, the graded formula $\Diamond_{\geq 2}\top$ can distinguish between the worlds w and w'.

In spite of this extra expressiveness, graded modal logic shares with the basic modal logic two properties: its complexity remains *PSPACE*-complete (Tobies, 2001b), and the tree model property still holds.

Graded modal logic behaves differently when interpreted on models with special frame properties. For instance, transitivity and euclidianness are two conditions that make the logic lose the tree model property. Consider the following formula interpreted on a transitive frame:

$$\Diamond_{\geq 2}(p \wedge \Diamond\neg p) \wedge \Diamond_{\leq 1}\neg p$$

Both successors of the evaluation point need a successor where $\neg p$ holds, but $\Diamond_{\leq 1}\neg p$ ensures that this successor is unique. Thus, models satisfying this formula need to have a diamond-like subgraph.

In (Kazakov and Pratt-Hartmann, 2009) it is shown that the complexity of graded modal logics over transitive or euclidean frames jumps to *NEXPTIME*-completeness. In the case of transitivity, tableaux system have been developed (Kaminski and Smolka, 2010c).

What about equivalence frames? We saw in Chapter 4 that they may lead to simpler satisfiability problems, so we may again get nice results here for graded modal logic.

Graded modal logic on equivalence frames is introduced as the logic $S5_n$ in Fine's article (1972). $S5_n$ is the logic obtained when the $\langle r\rangle$ operator is restricted to models where R_r is interpreted as an equivalence relation. Now, if R_r is the universal relation, then $\langle r\rangle_{\geq n}\varphi$ is obviously equivalent to the first-order formula $\exists^{\geq n}\varphi$. But a well known result (see, e.g. (Blackburn et al., 2001a)) establishes that the modal logic of the universal relation coincides with the modal logic obtained when we only require the accessibility relation to be an equivalence relation.

We can summarize this by saying that graded modal logic on equivalence frames is a logic that counts globally. It is in fact a syntactic variation of the one-variable fragment of first-order logic with counting, which, as we already mentioned, has an *NP*-complete satisfiability problem (Pratt-Hartmann, 2008). A satisfiability-preserving translation from $S5_n$ to FOL with one variable and counting is, essentially, given by:

$$\begin{array}{rcl} \mathsf{Tr}(\Diamond_{\geq n}\varphi) & = & \exists^{\geq n}x.Tr(\varphi) \\ \mathsf{Tr}(\Diamond_{\leq n}\varphi) & = & \exists^{\leq n}x.Tr(\varphi) \\ \mathsf{Tr}(p) & = & P(x) \end{array}$$

The fact that $S5_n$ enables global counting was also studied by van der Hoek and de Rijke (1993, 1995). In addition to providing axiomatizations, investigating normal forms, and establishing the complexity of the satisfiability problem for different logics with graded modalities, the authors propose these languages as a modal framework where some ideas from the Theory of Generalized Quantifiers (Westerståhl, 1989) could be investigated by means of modal tools.

Notice that in $S5_n$, counting operators can be nested. For example $\Diamond_{\geq 1}\Diamond_{\geq 2}p$ is a well formed formula, which it is actually equivalent to $\Diamond_{\geq 1}p$. But, as discussed by van der Hoek and de Rijke (1993), every $S5_n$ formula is equivalent to a formula where each counting operator appears under the scope of neither modal nor counting operators. The proof uses the fact that for any counting subformula σ appearing in a formula φ we have that the following is valid

$$\varphi[\sigma] \leftrightarrow (\sigma \rightarrow \varphi[\sigma/\top]) \wedge (\neg\sigma \rightarrow \varphi[\sigma/\bot])$$

Other operators with a global semantics, like the universal modality A or satisfiability operators i:, have the same property. Notice though, that the formula we obtain after extracting all counting operators can be exponentially larger. If we only require equi-satisfiability (and not equivalence), we can use the method of Areces and Gorín (2010) to obtain a formula which is only polynomially larger.

Since it is interpreted directly on equivalence frames, the language $S5_n$ does not enable mixing global counting with arbitrary accessibility relations and, *a fortiori*, graded modalities. However, a hybrid logic mixing these two types of counting is proposed by Kaminski et al. (2009b); Kaminski and Smolka (2010c). In these articles, a tableau system for hybrid logic with graded modalities and graded universal modalities, E_n and A_n, is given. Tableau rules for E_n and A_n are analog to those of graded modalities.

In the next section we are going to see that work has also been done in description logics in order to mix both types of counting.

5.1.3 Description logics

In the field of description logic (DL), we find work on both local and global counting, even though the former is much more developed. The DL equivalent of graded modalities is called *qualified number restrictions*. The corresponding syntax for these new concepts is:

$$\exists_{\geq n} R.C$$
$$\exists_{\leq n} R.C$$

or alternatively:

$$\geq n\ R.C$$
$$\leq n\ R.C$$

Semantically, there is no difference with graded modalities: concepts built with qualified number restrictions count on the successors. For example, if we extend the basic modal logic with graded modalities we obtain exactly the same concept language as $\mathcal{ALCQ}$. The complexity of this logic was investigated by Tobies (1999), who corrected some of the results of van der Hoek and de Rijke (1993, 1995), showing that satisfiability in $\mathcal{ALCQ}$ is *PSPACE*-complete. This result remains also when $\mathcal{ALCQ}$ is extended with inverse roles.

Qualifying number restrictions have been integrated in several tableaux calculi for expressive DL (see for instance (Horrocks and Sattler, 2007)), and have been successfully implemented in efficient automated provers such as FaCT++ (Motik et al., 2007), Pellet (Sirin et al., 2007), and Racer (Haarslev and Möller, 2001).

Qualifying number restrictions are useful for knowledge representation in many domains. For instance, they enable to express the following piece of knowledge:

$$Ethanol \;\sqsubseteq\; (=\ 8\ Contains\,.\,HydrogenAtom)$$

It has been noted that qualified number restrictions and relation of parthood between concepts create a jump in complexity for the satisfiability problem of this logic to *NEXPTIME*-completeness (Schröder and Pattinson, 2008b; Kazakov and Pratt-Hartmann, 2009). This happens since parthood is a transitive relation: for instance, if a school contains 4 classes and each class contains 20 students, then the school contains 80 students. As we saw with modal logics, graded modalities and transitive relations interact badly in that sense.

More dangerously, expressive knowledge bases often contain role inclusion axioms of the form $R \sqsubseteq S$, where R and S are roles, along with transitive roles (Rector and Horrocks, 1997). Horrocks et al. (1999a) showed that $\mathcal{SHIN}^+$, the DL with transitive roles, role inclusion, inverse roles and unqualified number restrictions on roles (i.e., of the form $(\geq n.R\top)$ and $(\leq n.R\top)$) is undecidable. As role inclusion is ubiquitous in knowledge bases, the solution to that problem has often been to prohibit number restrictions on transitive roles. Thus, a DL whose name contains $\mathcal{N}$ or $\mathcal{Q}$ instead of $\mathcal{N}^+$ and $\mathcal{Q}^+$ does *not* enable number restrictions on transitive roles. (Kazakov et al., 2007) show that decidability is regained when inverse roles are prohibited and when two transitive roles are never included in a same role. Another solution to regain decidability is to have number restrictions on transitive roles interpreted on trees (Schröder and Pattinson, 2008a), which is a reasonable restriction in some cases (Bittner and Donnelly, 2007).

So, counting is well-rooted into DL applications under the form of qualified number restrictions, that modal logic knows as graded modalities. Now, there is also a counterpart of global counting.

Baader et al. (1996) investigate *concept cardinality restrictions*. They introduce global counting restrictions under the form of TBox axioms $(\geq n.C)$ and $(\leq n.C)$. Thus, these constructs are not new constructs for concepts, but those are axioms that only appear at the top level of a knowledge base, along with other axioms such as general concept inclusions $(C \sqsubseteq D)$. The definition of satisfaction of these constructs by an interpretation I is given as follows:

$$\begin{array}{lll} I \text{ satisfies } (\geq n.C) & \text{iff} & |C^I| \geq n \\ I \text{ satisfies } (\leq n.C) & \text{iff} & |C^I| \leq n \end{array}$$

So here we find again our first-order counting on individuals of the domain. The authors remark that these counting axioms can express classical terminological axioms. Indeed, the general concept inclusion $\varphi \sqsubseteq \psi$ is satisfied in the model if the interpretation of φ is a subset of the interpretation of ψ, and this is the case exactly when $(\varphi \wedge \neg\psi \ \leq 0)$.

Baader et al. (1996) also introduce a tableau-based decision procedure for this language. The fact that concept cardinality restrictions only appear as TBox axioms makes the tableaux calculus simpler than (Kaminski et al., 2009b) for instance. This is because there are no tableaux rules to handle these axioms, but the tableaux branch is properly initialized according to "at least" constraints, and closure happens when an "at most" constraint is violated.

Tobies (2001a) gives a detailed complexity analysis of the satisfiability problem of several description logics with counting axioms. These DL involve *CBoxes*, that are similar to TBoxes but only contain axioms of the form $(\geq n.C)$ and $(\leq n.C)$, similarly to Baader et al. (1996).

Tobies obtains complexity results that depend on the encoding of numbers. Indeed, he uses a linear reduction of certain DL with CBoxes to first-order logic with two variables and counting (C^2). At that time, C^2 was known to be decidable and in (Pacholski et al., 1997) it was shown that satisfiability of C^2 was in *2NEXPTIME* under binary coding and *NEXPTIME*-complete under unary coding. This gave upper bounds on complexity. Lower bound was shown by encoding a domino system, which is a *NEXPTIME*-hard problem, into the studied DL. Thus the results obtained were:

Theorem 15. *(Tobies, 2001a)*

- *satisfiability of $\mathcal{ALCQI}$-CBoxes is NEXPTIME-hard under binary coding, and NEXPTIME-complete under unary coding*
- *satisfiability of $\mathcal{ALCQ}$-CBoxes is NEXPTIME-hard under binary coding and EXPTIME-complete under unary coding*

However, Pratt-Hartmann (2005) later shown that satisfiability of C^2 was *NEXPTIME*-complete even with binary coding of numbers. Thus, we can update

Tobies' results to:

Theorem 16.
- *satisfiability of $\mathcal{ALCQI}$-CBoxes is NEXPTIME-complete under unary and binary codings*
- *satisfiability of $\mathcal{ALCQ}$-CBoxes is NEXPTIME-complete under binary coding and EXPTIME-complete under unary coding*

Since these studies, concept cardinality restrictions have not received as much attention, and no notable implementation has been carried out for languages involving it.

5.2 Modal Logic with Counting

Now, we would like to study more in detail global counting. What language shall we use for this? In this section we will present global counting operators as an extension of the basic modal logic. In this way, we will still work with a decidable language, contrary to considering an extension of FOL. Moreover, we want to internalize global counting in the formulas of the language, in the same way as the universal modality is internalized in modal logic, so this is a conceptual advantage over DL and its separation between TBox and ABox. To finish with, this enables us to consider global counting inside of a language expressive enough to assert relations between individuals. Hence, this gives more expressiveness with regards to graded S5, which basically counts isolated individuals.

So let us use the basic modal logic and add counting operators such as:

$$Apple \geq 4.$$

The idea is that we want to make this counting operator the modal counterpart of first-order counting quantifiers. Let us thus introduce the formal syntax and semantics of the basic modal logic with counting $\mathcal{MLC}$:

Definition 34 (Syntax). *Let* $\mathsf{PROP} = \{p_1, p_2, \ldots\}$ *(the* propositional symbols*) and* $\mathsf{REL} = \{r_1, r_2, \ldots\}$ *(the* relational symbols*) be disjoint, countable infinite sets. The set* FORM *of formulas of* $\mathcal{MLC}$ *over signature* $\langle \mathsf{PROP}, \mathsf{REL} \rangle$ *is defined as:*

$$\varphi ::= \bot \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle r \rangle \varphi \mid \varphi \geq n$$

for $p \in \mathsf{PROP}$, $r \in \mathsf{REL}$, $\varphi, \varphi_1, \varphi_2 \in \mathsf{FORM}$ *and n an integer. Other Boolean and modal operators are defined as usual.*

We will call $\mathcal{PLC}$ the "propositional fragment," i.e., the fragment obtained by dropping $\langle r \rangle \varphi$. Let us now introduce the semantics.

Definition 35 (Semantics). *Given a signature* $\mathsf{Sig} = \langle \mathsf{PROP}, \mathsf{REL} \rangle$, *a* model *(or Kripke model) for* Sig *is a tuple* $\langle W, (R_r)_{r \in \mathsf{REL}}, V \rangle$, *satisfying the following conditions: (i)* $W \neq \emptyset$ *; (ii) each* R_r *is a binary relation on* W*; (iii)* $V : \mathsf{PROP} \to 2^W$ *is a labeling function.*

Given the model $\mathcal{M} = \langle W, (R_r)_{r\in\mathsf{REL}}, V\rangle$ and $w \in W$, the semantics for the different operators is defined as follows:

$$\begin{array}{lcl}
\mathcal{M}, w \models p & \iff & w \in V(p), \quad p \in \mathsf{PROP} \\
\mathcal{M}, w \models \neg\varphi & \iff & \mathcal{M}, w \not\models \varphi \\
\mathcal{M}, w \models \varphi \wedge \psi & \iff & \mathcal{M}, w \models \varphi \text{ and } \mathcal{M}, w \models \psi \\
\mathcal{M}, w \models \langle r\rangle\varphi & \iff & \text{there is } w' \text{ such that } R_r(w, w') \text{ and } \mathcal{M}, w' \models \varphi \\
\mathcal{M}, w \models \varphi \geq n & \iff & |\{v \mid \mathcal{M}, v \models \varphi\}| \geq n
\end{array}$$

Definition of satisfiability and local entailment in a Kripke model is as usual, see Chapter 2.

Moreover, the extension $\|\varphi\|^{\mathcal{M}}$ *of a formula φ in a model $\mathcal{M}$ is the set $\{w \mid \mathcal{M}, w \models \varphi\}$, and the* theory *of w in $\mathcal{M}$, notation $\mathsf{Th}^{\mathcal{M}}(w)$, is the set $\{\varphi \mid \mathcal{M}, w \models \varphi\}$. When the model $\mathcal{M}$ is clear from context we will drop the super-indexes. We will write $\mathcal{M}, w \equiv_{\mathcal{MLC}} \mathcal{M}', w'$ if $\mathsf{Th}^{\mathcal{M}}(w) = \mathsf{Th}^{\mathcal{M}'}(w')$.*

Numbers appearing in $\mathcal{MLC}$ formulas belong to $\mathbb{Z}$, and thus when n is negative, formulas of the shape $\varphi \geq n$ are trivially true.

5.2.1 Relation with other languages

The $\mathcal{MLC}$ language has interesting connections with other languages.

Universal and difference modalities, hybrid and description logics $\mathcal{MLC}$ can express the universal modality. This is because $A\varphi$ and $((\neg\varphi) = 0)$ are equivalent. As a consequence, $\mathcal{MLC}$ can express terminological axioms of Description Logics, since $\varphi \sqsubseteq \psi$ is equivalent to $\mathsf{A}(\varphi \rightarrow \psi)$.

Counting modalities can also express nominals by just stating $(p = 1)$ for p a propositional symbol, and hence they can be considered also as hybrid logics.

Putting this together, we get that the description logic $\mathcal{ALCO}$ is a fragment of $\mathcal{MLC}$. For instance, the following axiom (borrowed from (Sirin et al., 2004)):

$$RockFan \sqsubseteq Person \sqcap \exists hasIdol\{Elvis\}$$

Can be expressed in $\mathcal{MLC}$ (with syntactic shortcuts) as:

$$(Elvis = 1) \wedge \mathsf{A}\ (RockFan\ \rightarrow\ Person \wedge \langle hasIdol\rangle Elvis)$$

Since $\mathcal{MLC}$ can express nominals and the universal modality, $\mathcal{H}(\mathsf{E})$ is a sublogic of it.

$\mathcal{MLC}$ can even express the difference modality as $\mathsf{D}\varphi$ is equivalent to

$$(\varphi \rightarrow (\varphi \geq 2)) \wedge (\neg\varphi \rightarrow (\varphi \geq 1))$$

In plain words, either φ is true in the current world and then it has to be true in at least two places in the model, or φ is false in the current world and then it has to be true in at least one place of the model. In both cases this implies that φ is true elsewhere.

Hybrid graded logic with graded universal modality Finally, $\mathcal{MLC}$ is in its turn a fragment of the $\mathcal{SOQ}^+$ language, that is hybrid logic with graded modalities, global counting and transitivity, for which Kaminski and Smolka (2010c) introduce a decision procedure. The syntax of $\mathcal{SOQ}^+$ is given as follows:

$$\varphi ::= p \mid \neg p \mid x \mid \neg x \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle r \rangle_n \varphi \mid [r]_n \varphi \mid \mathsf{E}_n \varphi \mid \mathsf{A}_n \varphi$$

for $p \in \mathsf{PROP}$, $x \in \mathsf{NOM}$ and $r \in \mathsf{REL}$ where PROP, NOM and REL are disjoint countable sets of propositional, nominal and relation symbols. $\mathsf{E}_n\varphi$ and $\mathsf{A}_n\varphi$ are global graded modalities that behave like their successor-wise counterparts $\langle r \rangle_n \varphi$ and $[r]_n \varphi$.

Clearly, $\mathcal{MLC}$ is a syntactic variation of a fragment of $\mathcal{SOQ}^+$, as shown by the following translation:

$$\begin{array}{rcll}
\mathsf{Tr}(p) & = & p & \\
\mathsf{Tr}(\neg\varphi) & = & \dot{\neg}\mathsf{Tr}(\varphi) & \\
\mathsf{Tr}(\varphi \wedge \psi) & = & \mathsf{Tr}(\varphi)\dot{\wedge}\mathsf{Tr}(\psi) & \\
\mathsf{Tr}(\varphi \vee \psi) & = & \mathsf{Tr}(\varphi)\dot{\vee}\mathsf{Tr}(\psi) & \\
\mathsf{Tr}(\langle r \rangle \varphi) & = & \langle r \rangle_0 \mathsf{Tr}_\pi(\varphi) & \\
\mathsf{Tr}([r]\varphi) & = & [r]_0 \mathsf{Tr}_\pi(\varphi) & \\
\mathsf{Tr}(\varphi \geq n) & = & \mathsf{E}_{n-1} Tr(\varphi) & \text{if } n \geq 1 \\
\mathsf{Tr}(\varphi) & = & \top & \text{otherwise}
\end{array}$$

5.2.2 An explicit translation into $\mathcal{H}(\mathsf{E})$

In this section we introduce an equivalence-preserving translation between $\mathcal{MLC}$ and $\mathcal{H}(\mathsf{E})$. This translation involves using the shallow form of formulas of $\mathcal{MLC}$, then negation normal form (that we define), and the translation itself involves the introduction of new nominals. Consecutively, we show that a model satisfying such a translation can be stripped of its nominals and is a model of the initial $\mathcal{MLC}$ formula.

We introduce first the notion of *negation normal form* for $\mathcal{MLC}$.

Definition 36. *Given $\varphi \in \mathsf{FORM}$ the* negation normal form *of φ is obtained applying the following rules*

$$\begin{array}{rcl}
\neg\neg\varphi & \rightsquigarrow & \varphi \\
\neg(\varphi_1 \wedge \varphi_2) & \rightsquigarrow & (\neg\varphi_1) \vee (\neg\varphi_2) \\
\neg(\varphi_1 \vee \varphi_2) & \rightsquigarrow & (\neg\varphi_1) \wedge (\neg\varphi_2) \\
\neg\langle r \rangle \varphi & \rightsquigarrow & [r]\neg\varphi \\
\neg[r]\varphi & \rightsquigarrow & \langle r \rangle \neg\varphi \\
\neg(\varphi \geq n) & \rightsquigarrow & \varphi \leq (n-1) \\
\neg(\varphi \leq n) & \rightsquigarrow & \varphi \geq (n+1)
\end{array}$$

As we mentioned in Section 5.2.3, every formula in $\mathcal{MLC}$ is equivalent to a formula where each counting operators has been *extracted* and it appears under the scope of neither modal nor counting operators. Each $\mathcal{MLC}$ formula is equivalent to its extracted, negation normal form. Let $\mathcal{MLC}^{en}$ be set of extracted formulas of $\mathcal{MLC}$ in negation normal form. We now present a translation from $\mathcal{MLC}^{en}$ to $\mathcal{H}(\mathsf{E})$ formulas. Tr_π works by traversing formulas and adding new nominals so that counting claims are preserved (π is used to ensure that we always introduce new nominals, initially π is set to the empty string).

$$\begin{array}{rcl}
\mathsf{Tr}_\pi(p) & = & p \\
\mathsf{Tr}_\pi(\neg\varphi) & = & \neg\mathsf{Tr}_\pi(\varphi) \\
\mathsf{Tr}_\pi(\varphi \wedge \psi) & = & \mathsf{Tr}_{\pi 0}(\varphi) \wedge \mathsf{Tr}_{\pi 1}(\psi) \\
\mathsf{Tr}_\pi(\varphi \vee \psi) & = & \mathsf{Tr}_{\pi 0}(\varphi) \vee \mathsf{Tr}_{\pi 1}(\psi) \\
\mathsf{Tr}_\pi(\langle r\rangle\varphi) & = & \langle r\rangle\mathsf{Tr}_\pi(\varphi) \\
\mathsf{Tr}_\pi([r]\varphi) & = & [r]\mathsf{Tr}_\pi(\varphi) \\
\mathsf{Tr}_\pi(\varphi \geq n) & = & (\bigwedge_{1\leq i<j\leq n} x_i^\pi{:}\neg x_j^\pi) \wedge (\bigwedge_{1\leq i\leq n} x_i^\pi{:}\varphi) \\
\mathsf{Tr}_\pi(\varphi \leq n) & = & \mathsf{A}(\neg\varphi \vee \bigvee_{1\leq i\leq n} x_i^\pi)
\end{array}$$

in particular, if $n \leq 0$ then $\mathsf{Tr}_\pi(\varphi \geq n) = \top$ and $\mathsf{Tr}_\pi(\varphi \leq n) = \mathsf{A}(\neg\varphi)$.

Note that Tr does not traverse the subformulas on the right of the definitions as counting subformulas in $\mathcal{MLC}^{en}$ cannot be nested.

Let us call $\varphi^{\mathcal{H}_\pi}$ the formula obtained from the $\mathcal{MLC}$ formula φ by first extracting counting operators, transforming into negation normal form, and applying Tr_π; we write $\varphi^{\mathcal{H}}$ when π is the empty prefix.

Suppose now that $\mathcal{M}$ is a model satisfying $\varphi^{\mathcal{H}}$. We will show that counting has not been affected by the translation.

Definition 37. *We call a model $\mathcal{M}'$ a* naming extension *of $\mathcal{M}$ if it is a conservative extension of $\mathcal{M}$ for an extended language that only adds nominals.*

Theorem 17. *Let $\varphi \in \mathcal{MLC}$, and π an arbitrary prefix. There exists a naming extension $\mathcal{M}'$ of $\mathcal{M}$ such that $\mathcal{M}, w \models \varphi$ if and only if $\mathcal{M}', w \models \varphi^{\mathcal{H}_\pi}$.*

Proof. We can disregard the extraction and negation normal form steps of the transformation since they are equivalence preserving.
[$\Rightarrow$] The atomic, negation and modal connectors cases are immediate. For any model $\mathcal{M}$ let us represent as $\mathcal{M}{+}N$ any naming extension of $\mathcal{M}$ where N is the function that assigns nominals to elements of the domain of $\mathcal{M}$. Assume $\mathcal{M}, w \models \varphi_1 \wedge \varphi_2$, i.e., $\mathcal{M}, w \models \varphi_1$ and $\mathcal{M}, w \models \varphi_2$. By induction hypothesis $\mathcal{M}{+}N_1, w \models \varphi_1^{\mathcal{H}_{\pi 0}}$ and $\mathcal{M}{+}N_2, w \models \varphi_2^{\mathcal{H}_{\pi 1}}$. As N_1 and N_2 are defined on different nominals we can obtain $N = N_1 \cup N_2$ and we have $\mathcal{M}{+}N, w \models \varphi_1^{\mathcal{H}_{\pi 0}} \wedge \varphi_2^{\mathcal{H}_{\pi 1}}$, and hence $\mathcal{M}{+}N, w \models (\varphi_1 \wedge \varphi_2)^{\mathcal{H}_\pi}$. The case for $\varphi_1 \vee \varphi_2$ is handled similarly.

Assume $\mathcal{M}, w \models \varphi \geq n$, i.e., there exist n different states v_1 to v_n such that for all $1 \leq i \leq n$, $\mathcal{M}, v_i \models \varphi$. For any π, choose $N = \bigcup_{1\leq i\leq n}(x_i^\pi, v_i)$ to obtain $\mathcal{M}{+}N, w \models (\bigwedge_{1\leq i<j\leq n} x_i^\pi{:}\neg x_j^\pi) \wedge (\bigwedge_{1\leq i\leq n} x_i^\pi{:}\varphi)$ as needed.

Now, assume $\mathcal{M}, w \models \varphi \leq n$. Let v_1 to v_m ($m \leq n$) be all the states of $\mathcal{M}$ satisfying φ. For any π, introduce n nominals x_1^π to x_n^π and a mapping N such that for $1 \leq i \leq n$ there exists j, $1 \leq j \leq m$ such that $(x_i^\pi, v_j) \in N$ (two nominals can be true in the same state). Then $\mathcal{M}{+}N, u \models \neg\varphi \vee \bigvee_{1 \leq i \leq n} x_i$ for u an arbitrary state, and $\mathcal{M}{+}N, w \models \varphi^{\mathcal{H}}$.

[⇐] Let $\varphi \in \mathcal{MLC}$ and π an arbitrary prefix, and $\mathcal{M}'$ a naming extension of $\mathcal{M}$ such that $\mathcal{M}', w \models \varphi^{\mathcal{H}_\pi}$. If φ is a modal formula the implication is trivial.

Assume $\mathcal{M}', w \models (\varphi \geq n)^{\mathcal{H}_\pi}$. By definition $\mathcal{M}', w \models (\bigwedge_{1 \leq i < j \leq n} x_i^\pi{:}\neg x_j^\pi) \wedge (\bigwedge_{1 \leq i \leq n} x_i^\pi{:}\varphi)$. Since x_1^π to x_n^π are all true at different states $\mathcal{M}, w \models \varphi \geq n$.

Assume $\mathcal{M}', w \models (\varphi \leq n)^{\mathcal{H}(\pi)}$, i.e., $\mathcal{M}', w \models \mathsf{A}(\neg\varphi \vee \bigvee_{1 \leq i \leq n} x_i^\pi)$. Then an arbitrary u of $\mathcal{M}'$, $\mathcal{M}', u \models \neg\varphi \vee \bigvee_{1 \leq i < n} x_i^\pi$. Hence, either $\mathcal{M}', u \models \neg\varphi$ or $\mathcal{M}', u \models x_i^\pi$ for a given $1 \leq i \leq m$, i.e., $\{u\} = V(x_i^\pi)$ for $1 \leq i \leq m$. So there can not be more than n distinct states satisfying φ in $\mathcal{M}'$ and $\mathcal{M}, w \models \varphi \leq n$. □

Thus we can say that for a given $\mathcal{MLC}$ formula φ, a model of $\varphi^{\mathcal{H}}$ is a model of φ.

Since we introduced a terminating tableaux calculus for $\mathcal{H}(\mathsf{E})$ in Chapter 4 (equivalently we can use the calculus of Kaminski and Smolka (2009b)), we have concrete the tools to carry out inference in $\mathcal{MLC}$.

5.2.3 Expressive power

We established a few links between the expressive power of $\mathcal{MLC}$ and various logics in the previous section. On the other hand, we are going to see that the expressive powers of counting and graded modalities are incomparable. We will establish this in Theorem 20 using a suitable notion of bisimulation for $\mathcal{MLC}$ that we now introduce

Definition 38 (Bisimulation). *A* bisimulation *between two models $\mathcal{M} = \langle W, (R_r)_{r \in \mathsf{REL}}, V\rangle$ and $\mathcal{M}' = \langle W', (R'_r)_{r \in \mathsf{REL}}, V\rangle$ is a non-empty binary relation E between their domains (that is, $E \subseteq W \times W'$) such that whenever wEw' we have:*

Atomic harmony: *w and w' satisfy the same propositional symbols.*

Zig: *if $R_r wv$ then there exists a point $v' \in W'$ such that vEv' and $R'_r w'v'$.*

Zag: *if $R'_r w'v'$ then there exists a point $v \in W$ such that vEv' and $R_r wv$.*

Bijectivity: *E contains a bijection between W and W'.*

For two models $\mathcal{M}$ and $\mathcal{M}'$ and two elements w and w' in their respective domains, we write $\mathcal{M}, w \leftrightarroweq \mathcal{M}', w'$ if there exists a bisimulation between $\mathcal{M}, w$ and $\mathcal{M}', w'$ linking w and w'.

Theorem 18. *If $\mathcal{M}, w \leftrightarroweq \mathcal{M}', w'$ then $\mathcal{M}, w$ and $\mathcal{M}', w'$ satisfy the same formulas of $\mathcal{MLC}$.*

Proof. Assume there is a bisimulation E between $\mathcal{M}$ and $\mathcal{M}'$. Because of Atomic harmony, Zig and Zag, we know that E preserves all formulas of the basic modal language (Blackburn et al., 2001a). We only need to consider the counting operators.

Suppose then that $\varphi = (\psi \geq n)$ and let f be one bijection that by definition is contained in the bisimulation linking $\mathcal{M}$ and $\mathcal{M}'$. Assume that $\mathcal{M}, w \models (\psi \geq n)$. By inductive hypothesis $f(\|\psi\|^{\mathcal{M}}) \subseteq \|\psi^{\mathcal{M}'}\|$ and because f is a injective $|f(\|\psi\|^{\mathcal{M}})| \geq n$, hence $\mathcal{M}', w' \models (\psi \geq n)$. For the other direction, assume $\mathcal{M}', w' \models (\psi \geq n)$. Because f is a bijection we can consider $f^{-1}(\|\psi\|^{\mathcal{M}'})$ which has size greater than n , and by inductive hypothesis we know that it is a subset of $\|\psi\|^{\mathcal{M}}$. Hence $\mathcal{M}, w \models (\psi \geq n)$. The case for $\varphi = (\psi \leq n)$ is similar. □

As usual, the converse is not necessarily true but it holds on finite models.

Theorem 19. *Let $\mathcal{M} = \langle W, R, V \rangle$ and $\mathcal{M}' = \langle W', R', V' \rangle$ be two finite models and $(w, w') \in W \times W'$, $\mathcal{M}, w \leftrightarroweq \mathcal{M}', w'$ if and only if $\mathcal{M}, w \equiv_{\mathcal{MLC}} \mathcal{M}', w'$.*

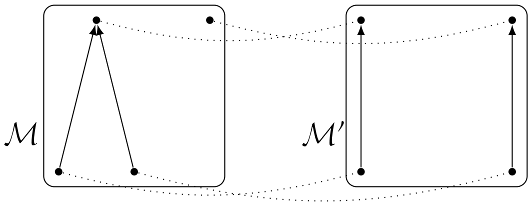
Proof. The implication from left to right is given by Theorem 18. For the other implication, we have to prove that $\equiv_{\mathcal{MLC}}$ is a bisimulation between $\mathcal{M}$ and $\mathcal{M}'$ that links w and w'. Atomic harmony, Zig and Zag are proved in the standard way (see (Blackburn et al., 2001a)). To prove that $\equiv_{\mathcal{MLC}}$ contains a bijection reason as follows.

Consider every pair of subsets (C, C'), $C \subseteq W$, $C' \subseteq W'$ such that for all $(a, b) \in C \times C'$, $\mathcal{M}, a \equiv_{\mathcal{MLC}} \mathcal{M}', b$. There is at least one such pair by hypothesis. Enumerate these pairs as $(C_1, C_1'), \ldots, (C_n, C_n')$ (as the model is finite there is only a finite number of them), and let $\Sigma_1, \ldots, \Sigma_n$ be such that $\Sigma_i = \mathsf{Th}(a)$ for some $a \in C_i \cup C_i'$ (by construction all elements in $C_i \cup C_i'$ satisfy the same formulas of $\mathcal{MLC}$). Now choose for each i, $\varphi_i \in \Sigma_i$ such that for all $j \neq i$, $\varphi_i \notin \Sigma_j$. Notice that $|C_i| = |\|\varphi_i\|^{\mathcal{M}}|$ and that $|C_i'| = |\|\varphi_i\|^{\mathcal{M}'}|$, we want to prove that $|C_i| = |C_i'|$. But by hypothesis $\mathcal{M}, w \equiv_{\mathcal{MLC}} \mathcal{M}', w'$, and then $\mathcal{M}, w \models \varphi_i = n$ if and only if $\mathcal{M}', w' \models \varphi_i = n$.

As C_i and C_i' have the same cardinality we can define an injective function $f : \bigcup C_i \to \bigcup C_i'$, such that for $a \in C_i, f(a) \in C_i'$. It only rests to prove that f is total and surjective.

Suppose there is $a \in W$ such that $a \notin \bigcup C_i$, then there is no element a' in W' such that $\mathcal{M}, a \equiv_{\mathcal{MLC}} \mathcal{M}', a'$. For each $a_i' \in W'$, let φ_i be a formula such that $\varphi_i \in \mathsf{Th}(a)$ but $\varphi_i \notin \mathsf{Th}(a')$. But then $\mathcal{M}, w \models (\bigwedge \varphi_i \geq 1)$ while $\mathcal{M}, w' \not\models (\bigwedge \varphi_i \geq 1)$ contradicting hypothesis. In a similar way we can prove that f is surjective. □

Notice that $\mathcal{MLC}$-bisimulations are not isomorphisms. The following two models, for example, are $\mathcal{MLC}$-bisimilar but not isomorphic.

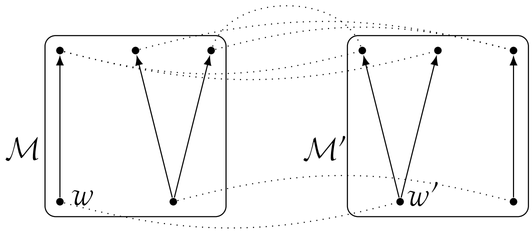


$\mathcal{M}$ and $\mathcal{M}'$ can be differentiated by the first order sentences $\exists x.\forall y.(\neg R(x,y) \wedge \neg R(y,x))$. But there is no $\mathcal{MLC}$ formula which is globally true in one model but false in the other. On the other hand, Westerståhl (1989) proves that every sentence of first-order logic with equality and only monadic propositional symbols is equivalent to the translation of a formula in $\mathcal{PLC}$.

We now return to the comparison of $\mathcal{MLC}$ and graded modalities.

Theorem 20. *The expressive power of counting modalities and graded modalities is incomparable (when interpreted on the set of all possible models).*

Proof. Consider the following two models $\mathcal{M}$ and $\mathcal{M}'$. It is not difficult to verify that the dotted arrows defines a $\mathcal{MLC}$-bisimulation.



$\mathcal{M}, w \not\models \langle r \rangle_{\geq 2} \top$ while $\mathcal{M}', w' \models \langle r \rangle_{\geq 2} \top$ while no formula of $\mathcal{MLC}$ can differentiate w and w'[1]. For the other direction, just consider a model with one state and another model with two states. Clearly, the models cannot be distinguished using graded modalities (as they can only count the number of successors) but the counting formula $(\top \leq 1)$ differentiates them. □

5.2.4 Complexity

The language $\mathcal{MLC}$ enables to easily fix the size of the model to any finite cardinality by setting

$$(\top = n)$$

This formula also shows that, if numbers are coded in binary, then neither $\mathcal{MLC}$ nor $\mathcal{PLC}$ has the polysize model property.

Proposition 6. *If numbers are coded in binary, then there are formulas in $\mathcal{PLC}$ (and hence also in $\mathcal{MLC}$) whose only models are exponentially larger.*

Hence, there is no guarantee that the complexity class of a logic involving counting is preserved when numbers are written in unary or in binary. In the unary case, one can easily get a complexity class result by using a simple translation into, for instance, hybrid logic, as we did in Section 5.2.1. Hence, we should take care of the encoding of numbers in any logic involving them.

[1] The proof goes through using the same models even if we add past operators to the language, as the bisimulation shown also satisfies the standard conditions Zig^{-1} and Zag^{-1} which preserve past operators (Blackburn et al., 2001a).

The complexity of the satisfiability problem for $\mathcal{MLC}$ and $\mathcal{PLC}$ have been studied in the literature. Let us call $\mathcal{L}^u$ and $\mathcal{L}^b$ the unary and binary coding, respectively, for either $\mathcal{MLC}$ or $\mathcal{PLC}$. Then, the previously established results are as follows.

Theorem 21. *1. $\mathcal{PLC}^u$-SAT and $\mathcal{PLC}^b$-SAT are NP-complete (Pratt-Hartmann, 2008).*

2. $\mathcal{MLC}^u$-SAT is EXPTIME-complete (Areces et al., 2000).

3. $\mathcal{MLC}^b$-SAT is EXPTIME-hard and in NEXPTIME (Tobies, 2001a; Pratt-Hartmann, 2005).

Proof. Hardness in all cases is clear, we only comment on the upper bounds. The proof of 1) is by a reduction to integer programming. The proof of 2) is by the previously shown polynomial satisfiability preserving translation into $\mathcal{H}(\mathsf{E})$. The proof of 3) is by a reduction to FOL with two variables and counting. □

Kazakov and Pratt-Hartmann (2009) gave a proof of the *NEXPTIME*-completeness of graded modal logic on transitive relations with binary coding of numbers is given. We conjecture that this proof might be adapted to show *NEXPTIME*-completeness of $\mathcal{MLC}$ also.

Let us now see what interesting problems we can carry out using these logics that can count. What can we possibly add to the already numerous inference tasks out there?

Among inference tasks in logic, we know very well *yes/no questions*, that is, satisfiability testing and validity testing. These two tasks are mutually reducible, granted the logic involved has the negation. In the domain of Description Logics, various inference tasks are directly reduced into satisfiability testing, as we saw in Chapter 2: knowledge base consistency, TBox consistency, instance checking, etc.

Another inference task is *retrieval*, that is, given a theory Γ, list all named individuals that make φ true, or more formally, obtain the set $\{i \mid \Gamma \models i{:}\varphi\}$. The retrieval task is typically reduced to repeated sat/unsat tasks: "find all names n such that $\Gamma \wedge (n{:}\neg\varphi)$ is unsatisfiable". There is work about optimizing this task, for instance by simplifying the knowledge base in order to efficiently eliminate candidates (Haarslev and Möller, 2008).

Now, what about *how many* questions, such as "How many apples are there on the table?". More formally, this question is "what is the n such as the cardinality of the extension of φ is n, if such a value exists ?". Of course such a question becomes interesting when asked in the context of some theory, so it should me more "given that Γ is true, what is the guaranteed cardinality of the extension of φ, if it exists ?".

Let us define formally this notion:

Definition 39. *Let $\Gamma \cup \{\varphi\}$ be a finite set of formulas in $\mathcal{MLC}$, we define the function $|\varphi|$ in Γ as follows*

$$|\varphi| \textit{ in } \Gamma = \begin{cases} n & \textit{if } \Gamma \models (\varphi = n) \textit{ and } \Gamma \textit{ consistent} \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

Let us take a very simple example. Let Γ be the set of formulas

$$\{\ (p = 2),\ (q = 3),\ (\neg(p \leftrightarrow \neg q) \leq 0)\ \}$$

In English this could be expressed by: there are exactly two p, exactly three q, and p and q are disjoint. In that case, we have that $|p \vee q|$ in Γ is equal to 5.

How do we compute such a value in general, given a set of formulas Γ and a formula φ? We could try to reduce this task to a list of satisfiability tests: "Are there 0 φ in all models where Γ hold?", "Is there 1 φ in all models where Γ hold?", "Are there 2 φ in all models where Γ hold?", etc. Each one of this questions can be reduced to testing the unsatisfiability of $\Gamma \cup \{\varphi \neq n\}$ so it seems possible. However, in the case where the answer to the initial question is "undefined", such a process may never stop. Thus such a series of test is not giving us a procedure for the counting task.

What we can do is doing this in two steps: first, check satisfiability of the theory Γ and build a model for it if, in the case it is satisfiable. Inside this model, count how many worlds make φ true. This is a candidate answer to the task. To verify that this candidate is really the answer, check satisfiability of $\Gamma \wedge (\varphi \neq n)$ with n being the candidate. If the answer "unsat", then we are sure n is the answer to the question "how many φ in Γ?".

We can now present the algorithm that carries out this task. Given P a decision procedure and model building procedure for $\mathcal{MLC}$, Γ a finite set of $\mathcal{MLC}$ formulas and φ a $\mathcal{MLC}$ formula:

```
if $P(\Gamma)$ returns UNSAT then
    return 'undefined'
else
    let n = $|\|\varphi\|^{\mathcal{M}}|$ for $\mathcal{M}$ a model returned by $P$
    if $P((\Gamma \wedge (\varphi \neq n))$ returns UNSAT then
        return n
    else
        return 'undefined'
    end if
end if
```

Theorem 22. *The algorithm above computes $|\varphi|$ in Γ.*

Although we used $\mathcal{MLC}$ as the logic to represent this task, any other logic with counting (FOL with counting, Description Logic or even Graded Logic) equipped with a sat solving and model building procedure would fit into this framework.

In the case of $\mathcal{MLC}$, the procedure we use to solve the counting task relies essentially on the satisfiability problem and on the model building task carried out by the previously mentioned decision procedures. To do so, we know we can either use the direct translation to hybrid logic with global graded modalities shown in Section 5.2.1 and use the decision procedure of Kaminski et al. (2009b); or use the translation to

$\mathcal{H}(\mathsf{E})$ also shown in Section 5.2.1 and use either the decision procedure presented in Chapter 4 or the one by Kaminski and Smolka (2009b). Both of these decision procedures have an implementation.

For a practical implementation of a calculus for any logic with counting, relying on arithmetic reasoning seems the right direction to take. Systems mixing tableaux procedures and arithmetic reasoning have been done for Description Logic in (Ohlbach and Koehler, 1999; Haarslev et al., 2001; Faddoul et al., 2008), The idea is to separate the counting constraints of the tableau and solve them with a constraint programming or an integer programming system. Thus, even with large cardinality constraints, unsatisfiable tableaux can be found efficiently and models can be represented in a compact way holding information of cardinality of its subparts in its complete version.

Chapter 6

HTab system description

HTab is a theorem prover for the hybrid logic $\mathcal{H}(:, \mathsf{E}, \mathsf{D}, \Diamond^-, \downarrow)$ with reflexive, transitive and symmetric modalities. It also has experimental support for injective and functional modalities, role inclusion and the transitive closure operator $\Diamond^*$. The logic experimentally supported is at least as expressive as the description logic $\mathcal{SHOIF}^+$.

The largest fragment for which termination of the program is ensured is $\mathcal{H}(:, \mathsf{E}, \mathsf{D}, \Diamond^-)$ with reflexive, transitive and symmetric modalities.

It has been developed as an implementation of the calculus described in Chapter 4. Its algorithm is a tableau calculus, thus it aims at deciding satisfiability of an input formula. This chapter explains the design and implementation of HTab. A few practical information about this program are listed here:

current version	1.5
license	GNU GPL
programmed in	Haskell (+ Glasgow Haskell Compiler extensions)
download page	`http://www.glyc.dc.uba.ar/intohylo/htab.php`
bug tracker	`http://code.google.com/p/intohylo/`

HTab depends on HyLoLib, a library containing modules to parse and manipulate hybrid logic formulas and models. This library was developed by Daniel Gorín, as part of his work on HyLoRes, a resolution-based theorem prover (Areces and Heguiabehere, 2001; Gorín, 2009). The library, along with HTab and HyLoRes, are available online under a free license, to encourage independent study and development.

6.1 Input formats

HTab can read two input formats: a simplified format, retained mainly for backwards compatibility and to ease comparison with other provers, and the default, more expressive, format.

Simplified format An input file has to start with the word `begin` to be recognized as being in the simple input format. Here is a sample file in the language $\mathcal{H}(:, \mathsf{E}, \Diamond^-)$:

```
begin
  A( N3 v [R1](-N1 v -[R2](-N2)));
  -N2 v -[R2](N2 v [R2](-N2 v [R1](N1)));
  [-R2](N3 v [R1](N1 v [-R2](N3 v [R2](N2))));
  N2 v [R1](-N3 v [-R1](N1 v -[R2](N1)));
  -N2 v -[R2](-N2 v [R1](-N2 v -[R3](-N1)))
end
```

In that format, words of the form N0, N1, ... are nominals, words of the form P0, P1, ... are propositional symbols and R0, R1, ... stand for relation symbols. The modalities A, E, B and D are available. The symbol - is used both for negations and to indicate converse relations.

It is possible to force all relations of the input formula to have desired properties with the command line parameters --allreflexive, --alltransitive, --allsymmetric, --allinjective and --allfunctional. Passing the switch -m MOD makes the prover output a model in the file MOD if the formulas are satisfiable.

Default format The default format is enabled when the input file does not start with begin. A file in that format has the following structure:

```
signature { ... }
theory { ... }
```

The simplest way of using this format is to use the automatic signature, which implies writing the theory formula as in the simplified format:

```
signature { automatic }

theory  {
  A( N3 v [R1](-N1 v -[R2](-N2)));
  -N2 v -[R2](N2 v [R2](-N2 v [R1](N1)));
  [-R2](N3 v [R1](N1 v [-R2](N3 v [R2](N2))));
  N2 v [R1](-N3 v [-R1](N1 v -[R2](N1)));
  -N2 v -[R2](-N2 v [R1](-N2 v -[R3](-N1)))
}
```

More interestingly, we can define which signature we want to use:

```
signature {
    propositions { tall, strong, naive }
    nominals     { alice, bob }
    relations    { love }
}

theory {
  alice: (strong & !tall & !naive);
  bob : (tall & !strong);
  (alice:bob) v bob:<love>alice;
}
```

Specifying propositions and nominals is just a matter of listing them. On the other hand, relations can be listed along with some extra properties, that are: `reflexive`, `symmetric`, `transitive`, `universal`, `difference`, and `functional`. Some properties take another relation as parameter `equals`, `inverseof`, `subsetof`, `tclosureof`, and `trclosureof`.

In the case of `subsetof` and `equals`, a list of relations can be given as parameter, and is interpreted as the union of its components. The following input file shows all these constructs (except difference):

```
signature {
    propositions
      { tall, strong, pretty, naive }
    nominals
      { alice, bob, jean, marie, unknown }
    relations
      { love,       lovedBy : {inverseof love},
        canManipulate : {trclosureof lovedBy},
        know : {reflexive},
        hasMet : {symmetric},
        U : {universal},
        fatherOf,   motherOf,
        parentOf : {equals {fatherOf,motherOf}},
        childOf : {inverseof parentOf, subsetof youngerThan},
        youngerThan : {transitive}
     }
}

theory {
 [U]((tall & strong) --> pretty);

 alice: ( strong  & !tall & !naive);
 bob: ( tall & !strong ) ;

 (alice:bob) v jean:<love>marie;

 bob:[lovedBy]naive;

 alice:<youngerThan>marie;
 marie:<youngerThan>bob;
 bob:<youngerThan>jean

 unknown:<parentOf>alice
}

query (satisfiable? , "out1") {
 alice:<canManipulate>jean
}

query (valid? , "out2") {
 bob:<hasMet>jean;
 marie:<hasMet>jean;
 jean:[hasMet]naive
}

query (retrieve , "retrieve1") {
 <youngerThan>jean
}
```

In the file above, we see three extra sections named query. Each query block specifies a reasoning task that is to be done against the background theory. Here, out1 and out2 are the files where the models will be written if they exist. retrieve1 is the file where the nominals making true the formula <youngerThan>Jean are written.

As of now, HTab handles queries as follows. Given a theory Γ and a query formula φ:

- a query of type satisfiable? is reduced to a satisfiability test of the formula $(\bigwedge\Gamma) \wedge \varphi$
- a query of type valid? asks for validity of the formula $(\bigwedge\Gamma) \to \varphi$, which is reduced to a satisfiability test of the formula $(\bigwedge\Gamma) \wedge \neg\varphi$
- a query of type retrieve asks for validity of the formulas $(\bigwedge\Gamma) \to a:\varphi$, where a is a nominal appearing in Γ, which is reduced to a succession of satisfiability tests of the formulas $(\bigwedge\Gamma) \wedge a:(\neg\varphi)$

There is currently no optimization of successive queries against the same background theory.

6.2 Internals

Rule application

The application of rules is made by managing a *todo* list which enables to define a strategy of rules application. For instance, one may want the rule ($\Diamond$) to be always applied before rule ($\vee$).

There are differences between the rules of the calculus as presented in Chapter 4 and how they are implemented. The rule ($\in$), which handles the equivalence classes of prefixes and nominals, is replaced in HTab by two rules. The first one, ($\in'$), is immediately applied on formulas of the form p, $\neg p$ when p is a propositional symbol (possibly a nominal). Indeed, there is no benefit in putting these atomic formulas in the todo list, when instead one can directly add them to the set of literals true at every node of the branch. The second one is the (merge) rule that handles merging of two equivalence classes, when a formula of the shape σa, with a being a nominal, is added to the branch. This rule is added to the todo list and can be processed with any priority with regards to the other rules (by default it has the greatest priority).

However, not all rules may be handled by the todo list, that is, some of them are immediately applied when the corresponding formula is added to the branch. This is the case of ($\Box$), for which it would be possible to choose a delayed application, but previous experiences have shown us that keeping track of combinations of premises of this rule could cost up to 20% of the running time of the prover on certain inputs.

Thus, the rules are divided into immediate and delayed ones:

- Immediate: ($\wedge$), ($\Box$), ($\Box^*$), (A), (B), ($\in'$), (inj), (fun), (tr), (re), (sy).
- Delayed: (merge), (:), (E), ($\Diamond$), (D), ($\downarrow$), ($\vee$), ($\Diamond^*$), ($\sqsubseteq$).

The default strategy of the delayed rules is the order given above.

Blocking and positive nominals

Two kinds of blocking can be enabled in HTab: anywhere blocking or pairwise blocking. If the input formula has the converse modality, the second one is activated. In all other cases the first one is used.

Blocking consists in preventing that the rules ($\Diamond$) and ($\Diamond^-$) be applied infinitely many times, by forbidding applications considered redundant. Redundancy is considered at the level of prefixes: a prefix is redundant when it can be replaced by another prefix. This contrasts with pattern-based blocking, a technique used in Spartacus (Götzmann et al., 2010), in which redundancy is considered at the level of accessibility relations.

Let $L(\sigma)$ be the set of local formulas true at a given prefix σ. We call local formulas all formulas of the shape p, $\neg p$, a, $\neg a$, $\Diamond_r\varphi$, $\Box_r\varphi$, with $p \in \mathsf{PROP}$, $a \in \mathsf{NOM}$.

Anywhere blocking works by forbidding application of ($\Diamond$) on a formula $\sigma\Diamond_r\varphi$ if there is a prefix $\tau < \sigma$ such that $L(\sigma) \subseteq L(\tau)$. Pairwise blocking forbids application of ($\Diamond$) and ($\Diamond^-$) on a formula $\sigma\Diamond_r^{(-)}$ if there are two prefixes τ_1, τ_2 such that $L(\tau_1) = L(\tau_2)$ and τ_1 and τ_2 are "ancestors" of σ in a chain of applications of ($\Diamond$) and ($\Diamond^-$).

These loop-checks involve comparing the set of formulas true at nodes and to conclude about the uselessness of expanding formulas of a given node if the information it contains is already present elsewhere. Since this notion of redundancy is only relevant at the local level (ie, we block two nodes if they are locally the same), it is useless to consider formulas whose main connector is :, A, and E. Moreover, we also do not consider formulas that are to be processed into other locally relevant formulas. This is the case for formulas $\wedge$ and $\vee$, but also D which is "converted" into local formulas.

But we can go further: we can ignore certain formulas that we know will never be constrained to be false at no prefix. This is the case for propositional symbols and nominals that always appear as true in the subformulas of the input formula (in negation normal form).

Moreover, this guarantees termination of the calculus involving the global difference operator B, since the new nominals introduced by one application of (B) are positive:

$$\frac{\sigma \mathsf{B}\varphi}{\sigma n, \gamma n \mid \gamma\varphi}\ (\mathsf{B}),\ n \text{ new},\ \gamma \text{ in the branch}$$

Thus, for instance with anywhere blocking, the presence of a formula σn introduced by (B) does not prevent the prefix σ from being blocked by a prefix τ for which $(L(\sigma) \setminus \{\tau\}) \subseteq L(\tau)$, since assuming n to be true at τ does not conflict with anything.

This treatment of positive nominals that can be ignored for loop-checks can be extended to nominals introduced by the rule (↓):

$$\frac{\sigma{\downarrow}x.\varphi}{\sigma n, \sigma\varphi[x \leftarrow n]}\ ({\downarrow}),\ n \text{ new}$$

If we know that the nominal x always appears positively in the subformulas of ${\downarrow}x.\varphi$, then we can treat the nominal n introduced by (↓) on $\sigma{\downarrow}x.\varphi$ as a positive nominal that can be ignored for blocking.

In some cases doing this makes HTab find out satisfiability of formula of $\mathcal{H}(:,{\downarrow})$ even when they contain a $\mathcal{U}{\downarrow}\mathcal{U}$ pattern (ten Cate and Franceschet, 2005). This is the case, for instance, for the formulas:

- $s : \Box{\downarrow}x.\Box s : \Diamond x \quad \wedge \quad s : \Diamond p \quad \wedge \quad s : \Diamond\neg p$
- $s : \Box{\downarrow}x.\Box s : \Diamond x$
- $\mathsf{A}{\downarrow}x\mathsf{A}\Diamond x$

This enables us to use the following trick: we do not implement the (B) rule as described above, but use in fact the following version:

$$\frac{\sigma \mathsf{B}\varphi}{\sigma{\downarrow}n.\mathsf{A}(n \vee \varphi)}\ (\mathsf{B}),\ n \text{ new}$$

Since n is positive in $\mathsf{A}(n \vee \varphi)$, it does not count for inclusion blocking, nor does $x \vee \varphi$ since the main connector is $(\vee)$.

Role inclusion and equality

The default input format of HTab enables one to specify roles inclusions and role equalities. For instance, writing the line

```
childOf : { subsetof youngerThan }}
```

in the signature corresponds to the inclusion $childOf \sqsubseteq youngerThan$.

Moreover, we allow definitions of the shape: $r \sqsubseteq \bigsqcup_i s_i$ (inclusion) and $r \equiv \bigsqcup_i s_i$ (equality). For instance,

```
parentOf : {equals {fatherOf,motherOf}}}
```

specifies the axiom $parentsOf \equiv (fatherOf \bigsqcup motherOf)$. A definition of the shape $r \equiv \bigsqcup_i s_i$ is rewritten as a list of definitions : $r \sqsubseteq \bigsqcup_i s_i$, $s_0 \sqsubseteq r$, ..., $s_n \sqsubseteq r$. This means that everything can be reduced to inclusion axioms of the form $r \sqsubseteq \bigsqcup_i s_i$.

Thus we need a tableau rule to handle this inclusion axioms, which is:

$$\frac{\sigma \Diamond_r \tau \ , \ r \sqsubseteq \bigsqcup_i s_i}{\sigma \Diamond_{s_0} \tau \mid \ldots \mid \sigma \Diamond_{s_n} \tau} \text{(role inclusion)}$$

However this feature does not guarantee termination in certain cases. Indeed, as shown in (Horrocks et al., 1999a), role inclusion axioms combined with transitive roles lead to undecidability. We can only guarantee that our calculus and implementation are sound and complete, but some entries may make the prover run indefinitely.

Injective and functional modalities

A relation is declared injective or functional in the signature of the input file, or by forcing all relations to be injective (resp. functional) with the argument `--allinjective` (resp. `--allfunctional`).

HTab uses two rules for these properties, and uses new positive nominals to ensure completeness and termination of the calculus:

$$\frac{\sigma_0 \Diamond_i \sigma_1, \sigma_0 \Diamond_i \sigma_2}{\sigma_1 n, \sigma_2 n} (f),\ i \in \mathcal{F},\ n \text{ new} \qquad \frac{\sigma_1 \Diamond_i \sigma_0, \sigma_2 \Diamond_i \sigma_0}{\sigma_1 n, \sigma_2 n} (f^-),\ i \in \mathcal{F}^-,\ n \text{ new}$$

Let us consider the (f) rule. If $\mathcal{R}_i$ functional, then all successors of a prefix σ_0 by an accessibility $\Diamond_i$ have to receive the same new nominal n. Thus σ_0 can only have one successor by $\Diamond_i$.

Again, termination in presence of injective and functional modalities is no guarantee. Moreover, the models returned by HTab do not enforce functionality and injectivity. Indeed, using these properties it is possible to define formulas whose only models are infinite. Intuitively, the models returned by HTab are folded representations of the real models.

Reflexive-Transitive closure modality

The reflexive-transitive closure modality comes from Propositional Dynamic Logic (PDL), whose satisfiability test is decidable with an *EXPTIME*-complete complexity.

Its semantic is defined by:

$$\mathcal{M}, w \models \Diamond_r^* \varphi \text{ iff } \text{ there exists } v \in W \text{ such that } (w, v) \in R_r^* \text{ and } \mathcal{M}, v \models \varphi$$

with R_r^* being the reflexive-transitive closure of the relation R_r.

As made clear by Sattler (1996) for instance, modal logic with the transitive closure modality is strictly more expressive than modal logic with a transitive relation.

Tableau-based decision procedures for modal logic with the transitive closure operator ($\mathcal{M}(*)$) have been proposed several times before. De Giacomo and Massacci (2000) presented a tableau calculus for $\mathcal{M}(*, \Diamond^-)$. However, it contains a cut-like rule that works as follows: for every node created from an ancestor by a relation E, it chooses between adding $\langle E^- \rangle \varphi$ and $\neg \langle E^- \rangle \varphi$ to it, for every φ in the Fisher-Ladner closure of the input formula. This kind of rules is of course very costly. However, a cut-free tableau procedure for $\mathcal{M}(*, \Diamond^-)$ has been recently proposed by Goré and Widmann (2010).

In parallel, Kaminski and Smolka (2010b) proposed a tableau calculus for $\mathcal{H}(*)$, and another one for $\mathcal{H}(*, \mathsf{D})$ (2010a).

HTab implements an experimental algorithm to handle transitive closure. No guarantee is given on its correctness, completeness, nor termination. This calculus is inspired from the one of De Giacomo and Massacci (2000), although it does not rely on a cut rule. The loop-check used in presence of the transitive closure modality is the chain-based twin blocking described in Chapter 4 for the calculus of $\mathcal{H}(\mathsf{D}, \Diamond^-)$.

The two extra tableau rules used are:

$$\frac{\sigma \Diamond^* \varphi}{\sigma \varphi \mid \sigma \neg \varphi, \sigma \Diamond \Diamond^* \varphi} (\Diamond^*) \qquad \frac{\sigma \Box^* \varphi}{\sigma \varphi, \;\; \sigma \Box \Box^* \varphi} (\Box^*)$$

Moreover, two other rules $(\Diamond^{-*})$ and $\Box^{-*}$ are defined accordingly. While handling $\Box^*$ does not pose a problem in terms of correctness and completeness of the calculus, $\Diamond^*$ is where things get complicated.

We call "eventuality" the formula φ inside a formula $\Diamond^* \varphi$. The rule $(\Diamond^*)$ tries to *fulfil* this eventuality at the prefix where the formula $\Diamond^* \varphi$ holds. If it fails, then it delays the eventuality, this is the right side of the conclusion of the rule. Here, when we delay an eventuality, we also add its negation on the right branch of the conclusion of $(\Diamond^*)$.

When an open branch is found, this is not enough to claim the formula satisfiable. We need to check that there are no unfulfilled eventualities. This book-keeping is done in HTab thanks to a data structure "Unfulfilled Eventualities" (UEV), which is simply an associative array, or map, from integers to a set of branching dependencies. Those are used to compute backjumping information (more on that in the next section).

In HTab, this structure is handled during the application of rule $(\Diamond^*)$ as follows:

- before the calculus starts, the input formula is rewritten such that every subformula of the shape $\Diamond^* \varphi$ is rewritten as $\Diamond^*(\mathsf{Nothing})\varphi$.
- when a formula $\sigma(\mathsf{Nothing})\langle r^* \rangle \varphi$ is added to a branch, we consult the structure UEV to get a new index for the eventuality φ at σ. The formula is then rewritten $\sigma(\mathsf{Just\ idx})\varphi$ and the couple (idx,deps), with deps being the branching dependencies of this subformula, is added to the UEV map.

- when an eventuality is realised, i.e., when the left branch of the ($\Diamond^*$) rule is chosen on a premise $\sigma(\mathsf{Just\ idx})\varphi$, the entry of key idx is deleted from the map UEV.
- when an eventuality is pushed away, i.e., when the right branch of the ($\Diamond^*$) rule is chosen, the formula $\sigma\Diamond\Diamond^*(\mathsf{Just\ idx})\varphi$ is added to the branch.

When an open branch is found, a test is performed to ensure that all eventualities have been fulfilled. It consists in checking that the mapping UEV is empty. If not, the branch is claimed closed, and, to calculate backjumping dependencies, the dependency set of this clash is the union of the dependencies of remaining entries of UEV.

The reason we add the negated eventuality to the right side of the conclusion in ($\Diamond^*$) is to prevent premature blocking, as shown in the example:

$$\neg p \quad \wedge \quad \Diamond^*(p \wedge \Box^- p \wedge \Box^- \Box^- p)$$

Let us call A the eventuality ($p \wedge \Box^- p \wedge \Box^- \Box^- p$). It is clear that to build a model for this formula, one has to realize the eventuality A far enough from the evaluation point where $\neg p$ holds. However, if the negation of A were not added in the right branch of applications of the ($\Diamond^*$) rule, blocking would happen too early and the formula would be claimed unsatisfiable by the tableau calculus.

As we have mentioned, the implementation is not only experimental and it is not based on a correct, complete and terminating calculus for $\mathcal{M}(*, \Diamond^-)$. Obtaining such a calculus will be a future work. For the moment we are carrying out testing with alternative provers (see (Hustadt and Schmidt, 2010)) to detect inconsistencies and bugs.

6.3 Optimisations

HTab includes a series of optimisations that are enabled by default. Let us detail them.

Semantic branching Semantic branching (Horrocks and Patel-Schneider, 1999) addresses one of the problems of the tableaux method, which is that the different branches of the tree might "overlap" (in terms of the possible models they represent). This leads to superposition of the search space explored by each branch.

The solution consists in adding to the second explored branch the negation of the formula added in the first branch — which is closed. The disjunction rule is replaced by:

$$\frac{\sigma(\varphi \vee \psi)}{\sigma\varphi \mid \sigma(\neg\varphi) \wedge \psi}\ (\textit{semantic branching})$$

Backjumping Backjumping is an optimisation that aims at reducing search space by replacing systematic one-level-up backtracking by dependency-directed backtracking. A simple example from (Horrocks and Patel-Schneider, 1999) is this formula:

$$(A_1 \vee B_1) \wedge (A_2 \vee B_2) \wedge \cdots \wedge (A_n \vee B_n) \wedge \Diamond(A \wedge B) \wedge \Box\neg A$$

Without backjumping we have to explore the whole search space created by the disjunctions on the left, while the causes of the clash — $\Diamond(A \wedge B)$ and $\Box\neg A$ — do not depend on them.

To be able to determine exactly up to which branching point we can backtrack, backjumping requires new information to be attached to prefixed formulas. We decorate each prefixed formula with its "dependency points" which are the branching point — i.e., the particular applications of the ($\vee$) rule — because of which the formula was generated. This information is then propagated to formulas obtained by the application of other rules: a formula depends on a particular branching point if it has been added to the branch at the moment of this particular application of the ($\vee$) rule, or if it has been added by the application of a rule where one of the premise formulas depends on this branching. The rules have to be adapted to propagate these dependencies, especially those that have several premise formulas like the ($\Box$) rule:

$$\frac{\sigma(d_1)\Box\varphi \text{ , } \sigma(d_2)\Diamond\tau}{\tau(d_1 \cup d_2)\varphi}\ (\Box)$$

In addition, we also need to ensure that the invariants that we implemented to account for the ($\in$) rule also propagate dependency information. As the aim of this rule is to copy formulas from one prefix to another according to the equivalence class they belong to, we choose to keep track of the dependencies of each equivalence class — i.e., the union of the dependencies of all the formulas that have contributed to the class. This is a quite radical solution, as it is not necessary to add the whole dependency set of a class to a copied formula to have a correct implementation of backjumping. The ideal solution would be to strictly keep track of the "path" that links two equivalent prefixes, instead of all contributions to the equivalence class. However, we found it simpler to attach dependencies to each equivalence class of prefixes.

While using backjumping, we can easily use the following heuristic: when we apply a rule of a certain type, we always choose the formulas whose earliest branching dependency is the smallest on the branch. This is done by maintaining each todo list by sorting formulas by smallest branching dependency. The aim is to boost the effect of backjumping. Indeed, tests have shown that while backjumping alone has a positive impact on performance, the previous heuristic enables HTab to behave an order of magnitude faster than before.

Unit Propagation and Eager Unit Propagation Tableau branching is a source of memory and time consumption and should be avoided whenever possible. Sometimes it is possible to reduce a disjunction before applying the ($\vee$) rule on it. If the branch contains the formulas σa and $\sigma(a \vee \varphi \vee \psi)$, and we were about to apply the rule $(\vee)$ on $\sigma(a \vee \varphi \vee \psi)$, we could take advantage of the situation and act as if the disjunction was only $\sigma(\varphi \vee \psi)$. This is what we call Unit Propagation (UP). If the disjunction is reduced to a single disjunct, then we have avoided branching, and just deterministically add the disjunct to the branch. If the disjunction is reduced to zero disjuncts, then we have a clash and need to backtrack.

An even more efficient approach is to not systematically taking the next formula in the $(\vee)$ todo-list and try to apply UP on it, but instead to choose the first formula of the todo-list on which UP can be applied. This is Eager Unit Propagation (UEP). The only drawback of EUP is that, for some input formulas, the todo-list is scanned too many times without result, making the global computation longer. However we have found that UEP worked in general well and better than simple UP, thus it is enabled by default.

Lazy Branching Another way to prevent branching from happening was introduced by Götzmann et al. (2010) and implemented in the hybrid theorem prover Spartacus. Lazy Branching (LB) is an optimization consisting is suspending disjunctions exploration as long as one of their conjuncts can be assumed true. We call *witnesses* formulas that we assume to be true in order to avoid exploring a disjunction.

For instance, the formula $\sigma(p \vee \Diamond\neg p)$ can be removed from the todo-list without provoking a branching in the tableau, if the witness p is added to the prefix σ.

The use of LB introduces a distinction between regular formulas that have been introduced by regular tableaux rules and witnesses formulas introduced by LB. Regular formulas are here to stay. If a conflicting formula is then added, the current branch clashes and backtrack is required to continue tableau exploration. Witness formulas are always on probation. If a regular formula that conflicts with a witness is added to the branch, then the regular formula stays, the witness goes away, and the disjunction attached to this witness is rescheduled (minus one disjunct if Unit Propagation is enabled).

In such a case where the disjunction is rescheduled, we still take advantage that branching happens lower in the tableau tree, and the lower the better, because this may factorize redundant section of the tableau.

Now, not all formulas can be witnesses. A witness has to be an assumption local to a prefix. If not, if would be difficult to find whether there are witnesses to revoke when a regular formula is added to the branch. So the witnesses possible are positive or negative propositional symbols, and negative nominals. Positive nominals cannot be witnesses because their addition to a prefix creates a merge of equivalence class between the concerned prefix and another prefix where the nominal holds (and the calculus guarantees there is always another one). Another possible witness is $\Box\bot$. This witness is revoked as soon as a formula $\Diamond\varphi$ is added to the concerned prefix.

The distinction between regular formulas and witnesses goes away at model building time: all formulas are taken as true, and need to appear in the model. This implies, at model building time, that prefixes with witnesses cannot be “represented” by other prefixes. To do that, and still use the model building definition presented in Chapter 4, we want to modify Inclusion Blocking (IB) so that the test of inclusion is done on the complete set of formulas assumed to be true at a given prefix, regular or witnesses.

But there is a problem in that case: termination is no longer guaranteed if IB takes witnesses into account. Indeed, in some situations, infinitely many prefixes may successively get the privilege to expand their $\Diamond$ formulas by being in a transient state T of local formulas, and then revoking some of their witnesses so that they are no longer in state T. Thus, future prefixes may be in that transient state T again and get the privilege to expand their diamond formulas. In other terms, the pigeonhole principle, on which IB relies to ensure termination of the calculus, does no longer hold. So it is crucial that IB never takes witnesses into account. But then, IB has to block prefixes for which no witness can be present.

Put it in another way, for a tableau calculus with Inclusion Blocking to remain adequate, Lazy Branching can only be applied on the prefixes whom we know the nominal urfather will remain in the model built. These can only be the initial prefixes of the tableau: the prefix at which the input formula holds, and the prefixes at which each nominal of the input formula hold. So in HTab, Lazy Branching only happens on these prefixes.

What happens in Spartacus? Spartacus uses Pattern-Based Blocking (PBB), and also every node of an open tableau branch appears in the model, since model construction does no filtration on nodes and only add missing links between nodes. PBB does not block a node n from expanding its $\Diamond$ formulas on the basis that the set of local formulas of n is included in the set of local formulas of another node. Instead, a formula of the form $\sigma\Diamond_r\varphi$ is disallowed to be expanded only when its pattern $P_\Theta(\sigma\Diamond_r\varphi) \equiv_{def} \{\Diamond_r\varphi\} \cup \{\Box_r\psi \mid \sigma\Box_r\psi \in \Theta\}$ is included in the pattern of another diamond formulas that has already been expanded.

Thus, if Lazy Branching is enabled in the calculus, PBB does not even consider the witnesses added by this technique. They do not interfere with blocking, thus they do not interfere with termination. In other terms, the power of Lazy Branching is fully realized in a calculus with PBB as opposed to a calculus with IB. The addition of PBB to HTab is another point for us to investigate.

However, two positive remarks on the current situation. First, although LB does not generally greatly improve the performance of HTab, in some cases already improves the performance of HTab, it does dramatically in some specific cases. For instance, a formula of the form $\mathsf{A}(\wedge_{1\leq i\leq n}(p_i \rightarrow \Diamond q_i))$, with $p_i \in \mathsf{PROP}\setminus\mathsf{NOM}$, would be very slow to process without LB for great values of n. Second, we concluded that LB dealt badly with loop-checks that rely on local formulas of prefixes. Although we can avoid this for the logic $\mathcal{H}(\mathsf{D})$ by using PBB, we still have to rely on such a loop-check for $\mathcal{H}(\mathsf{D},\Diamond^-)$. So in this last case, LB has to be restrained on a fixed subset of prefixes anyway.

6.4 Other systems and evaluation

In (Hoffmann and Areces, 2009) we evaluated an older version of HTab against the theorem provers HyLoTab and HyLoRes, who were the only other hybrid logics provers available at that time. We found out that HTab was in most cases more performant than these tools. However, since then was released the prover Spartacus (Götzmann et al., 2010), which handles the hybrid logic $\mathcal{H}(;,\mathsf{E})$ and is generally more efficient than HTab. At the time of writing this document, Spartacus is no longer developed, but is in a stable and finished state. This is why we will now rather focus on evaluation HTab against Spartacus.

We should also mention the systems Herod and Pilate, developed by Mayer and Cerrito (2010). These system handle the basic hybrid logic $\mathcal{H}(:)$ and are based on a calculus that aims at handling nominal equality in an efficient way. However, since these provers for $\mathcal{H}(:)$ are still undergoing development, we preferred not to include them in our comparisons.

Since a few years, the extensible tableaux-based prover LoTREC (Fariñas Del Cerro et al., 2001; Gasquet et al., 2005) is developed. Based on graph rewriting, it enables users to define their tableaux system for the logic of their choice in a high-level language that prevents them from getting into the dirty details of programming. It is this a platform to experiment with tableaux systems. LoTREC comes with a set of predefined calculi for various logic, and it was extended by Said (2010) to handle the basic hybrid logic $\mathcal{H}(:)$. This prover is aimed at being usable and extendable, and is not oriented towards performance or generic execution. For this reason we will also not compare it with HTab on grounds of performance.

In the next chapter, we will present the benchmark system we used during the development of HTab. This will enable us to present the tests we ran to verify the positive effects of the previously mentioned optimizations on performance of our prover. We will also present tests comparing HTab with other theorem provers for hybrid and description logics.

Chapter 7

Benchmarking

Benchmarking is the act of running a computer program in order to assess the performance of another program. Computer programs that run through this kind of examination are typically compilers, databases management systems, web browsers, etc. In fact, any library performing a precise task may benefit from such an examination.

Benchmarks are also an important part of the toolbox used by theorem provers developers. Usually, every article describing a new system comes with its benchmarks section, where the system is compared to existing provers.

We benchmark theorem provers for several purposes:

- to check whether some changes introduced in a piece of software has a positive impact on performance. In the case of theorem provers this is particularly useful when it comes to optimizations and heuristics.
- to evaluate the relative performance of different provers and put their behaviours into perspective.
- as a side-effect, to communicate with other theorem provers developers (this is why benchmarks appear in system descriptions)
- to help debugging provers, by comparing several provers and checking whether their answers are consistent for a given input.

A benchmark program involves collecting data to feed the benchmarked program, and summarizing and visualizing the results. Because programmers need to trust their benchmarks, they have to be correct and informative. Visual summarization also plays an important role. Presentation of information can range from comparing the output of two runs of the `time` command, to displaying a graph showing times of repeated runs of provers with regards to characteristics of its input, as we will see later.

Since in the end, it is the user (or the programmer) that takes decisions because of benchmarks results, the way information is presented has an undeniable influence. The information obtained in the benchmark must be properly conveyed. So, not only correctness of the benchmark is important, but also the way it represents information, hopefully in an easily interpretable and non-misleading way.

In this chapter we will present the benchmarks we use along with some general guidelines on benchmarks relevance, we then discuss some possible improvements and also mention possible new ways of testing modal provers.

7.1 Covering the input space

For a benchmark result to be significative, it has to be produced from a representative input set. One may want two kinds of representative input: either an input covering completely the set of possible input formulas, or an input covering a subset corresponding to a certain use.

In the case of propositional logic, things are quite easy. The following is an example of the input format of solvers that run the SAT-Race 2010[1] is:

```
p cnf 5 3
1 -5 4 0
-1 5 4 2 0
-3 -4 0
```

The first line indicate the number of variables (5) and the number of clauses (3) appearing in the file. Then each line represents a clause, that is a list of numbers, with a minus sign indicating negated literals. Thus the formula above is in fact

$$(1 \vee \neg 5 \vee 4) \wedge (\neg 1 \vee 5 \vee 4 \vee 2) \wedge (\neg 3 \vee \neg 4)$$

It is clear that the space of possible inputs is quite simple to cover. Let us choose to fix the number of disjuncts per clause to any number equal or greater than 3, and fix the number of propositional symbols to n. Now, by varying the number of clauses c, and thus the ratio n/c, we can generate instances of formulas ranging from mostly satisfiable to mostly unsatisfiable. This enables us to observe the behaviour of a prover on formulas that should require different exploration of the search space in order to be proven satisfiable or unsatisfiable.

This organisational simplicity is reflected in the now traditional SAT competition. It involves three kinds of formulas: random formulas (van Gelder, 1993), handcrafted formulas whose satisfiability is known beforehand, and formulas coming from real-world applications. With these three batches, all presented in the same format, it is assumed that provers are thoroughly tested.

Now, the case of first-order logic is trickier. How can one be sure to aptly cover enough input space to check ones's prover? The answer is that it is very hard to test thoroughly a first-order prover. Because of quantification, the presence of constants and variables, and the presence of relational symbols, it is hard to obtain a test in which a unique parameter can be modulated so as to generate the same sat-to-unsat progression.

So, what is done for FOL, is a selection of test formulas that test special sets of features. For instance, if one wants to test their prover for equality, one has to prepare formulas exactly for that. This is what TPTP (Thousands of Problems for Theorem Provers) is about (Sutcliffe et al., 1994).

Modal logics (including hybrid and description logics) fall between these two cases. Fixing a certain signature, it is certainly harder to cover all input space than in the

[1] http://baldur.iti.uka.de/sat-race-2010/

propositional case, since modal depth comes into play. However, since decidable modal logics usually do not involve explicit variable binding, the syntactic possibilities are greatly reduced.

A great amount of work has been done in order to find good test suites for modal logic provers.

Heuerding and Schwendimann (1996) introduced a series of tests for modal provers, guaranteed to be hard enough so that provers run enough time and can be sufficiently evaluated. The formulas generated by their software are of known satisfiability with regards to arbitrary models (K), reflexive models (KT) and transitive models (S4). The authors give, for each of these categories, several recipes to build benchmark formulas. The recipes are not random-based.

Giunchiglia and Sebastiani (1996) introduced KSAT, a propositional sat-based description logic prover, for which they assessed the performance with regards to other DL provers by using a random formula generator. The basic idea of this generator is to generate 3SAT clauses, and to sometimes replace propositional symbols by modal subformulas, and then recursing into these subformulas until a specified depth limit is reached. However, Hustadt and Schmidt (1997) pointed out that this benchmarking suite generated too many trivial instances, that were directly decided by sat-based provers in linear time. The proportion was from one quarter to three quarter of the generated formulas, thus leading the authors to wrong conclusions. Giunchiglia et al. (1998) integrated this criticism and fixed the benchmarking suite to evaluate KSAT again.

Massacci (1999) introduced the TANCS test suite (TAbleaux Non-Classical Modal System Comparisons), also based on random formula generation, but building upon the previous experience of Giunchiglia and Sebastiani (1996) and (Hustadt and Schmidt, 1997). Most importantly, their test suite attempts to evaluate handling of the modal aspect of formulas, as opposed to handling the propositional aspect. This is done by designing a series of subtests, coming from the encoding of various logics into modal logic, and making sure that propositionally trivial instances can not be generated.

Horrocks et al. (2000); Patel-Schneider and Sebastiani (2003) build upon all the previous proposed benchmarking suites by analyzing their limits and proposing more trustable suites. In particular they present a way to generalize the test suite used for KSAT to arbitrary shapes of conjuncts, while preventing as much as possible that the generated formulas depend too much on their pure propositional satisfiability.

Finally, Gardiner et al. (2006a,b) continued this reflexion by proposing benchmarks for very expressive description logics, based on preexisting ontologies. They compiled these ontologies into a library of benchmarking problems for testing state-of-the-art DL provers. Moreover, their tests not only consider TBox satisfiability testing, but also class hierarchy building. Correctness of the results is tested by comparing the outputs of several provers run on the same problem. The author note that their benchmarking suite helped us find bugs in DL provers.

7.2 Presentation of GridTest and hGen

The benchmark program we used to test the algorithms and implementations presented in this thesis is called GridTest. Although its present distributed form was described in (Areces et al., 2009), it started its life in 2004 as a series of scripts used to evaluate the resolution-based hybrid prover HyLoRes against state-of-the-art modal and description logic reasoners, running on a single machine (Areces and Gorín, 2005).

The basic idea of GridTest is to test theorem provers using randomly generated formulas. The objective is to design tests according to a set of fixed parameters, like modal depth or number of propositional symbols, and try to cover the a sat-to-unsat repartition of formulas by varying the number of clauses contained in the formulas.

This tool can be used to run tests locally, in a single computer, or in a computer grid. It generates reports which, among others, includes graphs for time comparison. It can compile statistics provided by the provers (e.g., running time, number of applications of a particular rule, open/closed branches, etc.) and produce graphs generated using GnuPlot. Even if the prover does not provide any kind of statistics, GridTest will use the `time` command (available in all POSIX-conformant operating systems) to obtain running times to plot in the final report.

GridTest has been designed for automatizing tests as those described in (Patel-Schneider and Sebastiani, 2003). That is, we specify a certain set of propositional symbols, nominals and relation symbols that can appear in formulas, and use a random generator of formulas in conjunctive normal form (CNF) to obtain batches of formulas with an increasing number of conjunctions. Because they are conjunctive, clauses act as constraints on satisfiability of formulas: CNF formulas with few clauses will tend to be satisfiable and CNF formulas with lots of clauses tend to be unsatisfiable. We then display, for prover involved, the median time on each batch of formulas. On this median time graph also appear the 25% and 75% percentile times, so as to visualize better the repartition of times in a batch. Hence, each prover is represented by a line that shows its behaviour against bigger and bigger formulas. In this way, we can visualise the behaviour of provers on formulas whose satisfiability is easy to prove, on formulas whose unsatisfiability is easy to prove, and in between, on formulas whose satisfiability is the most random and hard to prove. This is why we aim at having a bell-like curve, so that the global performance of a given prover can be visualised.

Indeed, as a formula gets more and more conjuncts, its satisfiability constraints get more numerous, making it go from most probably satisfiable to most probably unsatisfiable. In between, formulas tend to be in an area of maximum uncertainty, thus being difficult for most provers, regardless of whether they are naturally biased towards satisfiable or unsatisfiable formulas. This is a way to exhibit interesting performance difference between provers. In the case of 3SAT, this phenomenon has been called the *transition phase* (Gent and Walsh, 1994). For modal logics, or more generally, *PSPACE* problems (as opposed to *NP* problems like 3SAT), this phenomenon has been observed but less accurately studied (Gent and Walsh, 1999).

At the beginning of a benchmark run, GridTest invokes a random formula generator so as to generate a sequence of sets of formulas. For instance, one can specify sets of formulas of size 10, 15, 20, . . . until 100. Given a specific test, one wants to tune these parameters so as to exhibit a "phase transition" behaviour. All sets have a fixed given size. Properly adjusting with last parameter enables us to have a tradeoff between total time of execution of the benchmark and reliability of its result. Properly adjusting the range parameter enables us to chose the "area" of formulas that are interesting to us.

We use GridTest mostly to test theorem provers for hybrid logics and hence the current framework uses hGen as its random formula generator (Areces and Heguiabehere, 2003). hGen generates formulas in a conjunctive normal form: each formula is a conjunction of disjunctive clauses. Initially, hGen supported the hybrid logic $\mathcal{H}(:, \mathsf{E}, \downarrow)$. It builds upon previous analysis of random modal formulas generators (e.g., (Patel-Schneider and Sebastiani, 2003)), and in particular avoids generating trivial subformulas specific to hybrid logic. For instance, subformulas of the shape $n:(n \vee \varphi)$ or $\downarrow x.(x \vee \varphi)$ are never generated.

hGen can now generate random formulas in the hybrid language $\mathcal{H}(:, \downarrow, \mathsf{E}, \mathsf{D}, \Diamond^{-})$. But because we are interested in the comparison of provers for different logics (e.g., description and first order logics) the framework is designed to properly handle translations between the output format of the random generator and the input format of the different provers (in such case, translation time is displayed separated from actual prover time). A number of translations from the output format of hGen to the input format of different provers is provided with the source code (e.g., the TPTP format for first-order provers, the standard input format for description logic provers, etc.), together with the drivers for different provers (e.g., E, SPASS, Bliksem, Vampire, Racer, FaCT++, HTab, HyLoRes, etc.).

hGen itself is customisable and can generate formulas according to the following parameters:

- the number of propositional symbols
- the number of nominals
- the number of state variables
- the number of relations
- the maximum nesting of all modal connectors, called the "maximal depth"
- the maximal nesting of diamonds, satisfaction operators, down-arrow binders, inverse modalities, universal modalities and difference modalities

Moreover one can set the frequency of propositional symbols, nominals, state variables, diamond formulas, satisfaction formulas, down-arrow formulas, converse diamond formulas, universal formulas and difference formulas. A frequency of zero means that the given logical connector never appears in the formula, thus hGen can generate formulas of the basic modal logic, for instance.

To summarize, here is how a typical benchmark unfolds: i) generate random formulas $\varphi_1 \ldots \varphi_n$ where φ_i has exactly i conjunctions and the rest of the parameters are fixed, ii) run provers $p_1 \ldots p_k$ over each of the n random formulas, using a fixed time limit per formula, iii) collect data of interest about each run (execution time, answer, number of rules fired if available, etc.) and plot it for comparison. Of course, there are no precise statistical foundations for this experiment, but by repeating it a sufficiently large number of times (or, equivalently, using batches of formulas sufficiently large for each data point) and using an estimation on the sampled data (e.g., average, median, etc.) statistically relevant results can be obtained.

In order to run on a single machine, GridTest requires a python interpreter and some typical POSIX tools (`bash`, `time`, `tar`, etc.). Its output is a collection of GnuPlot and LaTeX scripts that are automatically compiled into a PostScript file reporting the results. These requirements are fairly typical and are available on almost every platform.

7.3 Splitting running time with a computer grid

The methodology presented in the previous section is simple, but it presents a clear drawback: even for tests of a moderate size, if we generate non-trivial formulas and allow each prover to run for a reasonable amount of time, the total running time on a single computer can become enormous. Tests with running times measured in days (or weeks) become common. This is especially true if some of the provers involved in the test tend to time out often. If we are interested in using this form of testing as part of the development process of a prover, rapid availability of the results is crucial.

The good news is that because of the nature of the tests, we are not obliged to run all executions of provers serially in the same computer. We can, as well, obtain statistical relevance by distributing the tests on a computer cluster: each machine runs the complete tests on batches of smaller size and the data is pulled together for statistical analysis when all the runs are completed.

Concretely, instead of running a test with batches of size b on a single computer, we could alternatively run tests on n different computers, each having to process a batch of size b/n, obtaining a linear reduction on the time required.

Although large computer clusters are not ubiquitous, the recent emergence of *grid computing* technologies is giving researchers access to a very large number of computing resources for bounded periods of time. In this scenario, it is not unreasonable to assume the simultaneous availability of such a number of computers.

Even if the grid is composed of heterogeneous machines (different processors, clock speeds, cache memory sizes, etc.), qualitative result (i.e., the relative performance of the provers under evaluation) would not be affected. On the contrary, it can even make the obtained results more trustworthy. Indeed, the danger of running a benchmark on a single hardware and software configuration is that some external parameter may influence the results. For instance in (Mytkowicz et al., 2009), it is shown that benchmarks comparing -O2 and -O3 optimizations of the gcc compiler can be biased by external parameters like the size of the UNIX environment, and linking order. Depending on the situation, one can be lead to think that using -O3 instead of -O2 yields a difference going from -10% to +10% of performance, thus drawing incorrect conclusions.

Because of this, one desired future evolution of GridTest is the ability to be run across any set of remote computers accessible by ssh. A good side effect is that it may give a second life to unused computers that tend to accumulate in computer science laboratories over time.

As of now, unlike the POSIX standard for operating systems, there is no standard batch scheduling mechanism for computer clusters or grids. GridTest currently supports only one backend, namely the OAR (The OAR team, 2010) batch scheduler, to distribute the test on a computer cluster. We designed GridTest so as to use only very basic services that most batch schedulers should provide, but still, porting it to other systems could be the most difficult challenge when trying to use GridTest elsewhere.

7.4 GridTest in action

Given the high number of possible dimensions in modal testing, it is important to clearly know what a test is supposed to prove, what hypothesis one wants to confirm or infirm. We will see GridTest in action with two benchmarks, each one having a different purpose that will be explained. For more information about the requirements and configuration of GridTest, see (Areces et al., 2009).

A general remark on the benchmarks that follow. Each one of them is dedicated to a specific measurement, and is designed with fixed initial parameters, like the number of propositional variables and nominals, and maximum modal depth. However, once these parameters are fixed, we played on the bounds of the size of the generated input formulas so that each test covers an input space from mostly satisfiable to mostly unsatisfiable formulas. This is why each test has different ranges on the size of formulas.

Also, all tests are done with only one accessibility relation. Increasing the number of relations only makes the formulas less constrained, and there are no interesting interactions or optimizations to consider – yet – for hybrid provers with regard to multiple relations. Fixing this parameter helps us to focus on the other ones.

Unless precised, the software versions we used was HTab 1.5.4, Spartacus 1.1.3 and FaCT++ 1.5.0 with default settings.

HTab optimizations performance on modal and hybrid logic As promised in the previous chapter, we are going to verify the role of the implemented optimizations on the performance of HTab. The benchmark that follows will involve:

- `htab`, that is HTab with default settings
- `htab L`, default settings without Lazy Branching
- `htab S`, default settings without Semantic Branching
- `htab B`, default settings without Backjumping
- `htab E`, default settings without Eager Unit Propagation but with Unit Propagation

- `htab U`, default settings without any form of Unit Propagation

The methodology we use is to disable one optimization at a time to see its effect. It is not an ideal way of assessing effects of individual optimizations, since these can interact with each others, but there is only so much space in a graph, so comparing 6 versions of the prover is already a good start.

The first benchmark is run on modal formulas, that is, without nominals nor universal modality. These formulas contain 8 propositional symbols, no nominals, a maximum modal depth of 2. Clauses of these formulas look as follows:

```
-P1 v [R1](-P7 v [R1](-P5 v -P2));
P6 v [R1](P6 v [R1](-P8 v P3));
-P2 v [R1](P6 v -[R1](-P5 v -P6));
-P3 v -[R1](-P8 v -[R1](-P8 v P2));
P5 v -[R1](-P6 v [R1](P7 v P6));
```

The test was run on batches of size $20, 60, \ldots, 400$ with a timeout of 90 seconds. Figure 7.1 show the sat/unsat repartition of formulas, as assessed by HTab with all optimizations. Figure 7.2 show the median time results. We can see that each optimization benefits to HTab, which the exception of Lazy Branching, which does not change its performance on this test. Backjumping, Unit Propagation, Eager Unit Propagation and Semantic Branching bring significant improvements to performance, although their effect differs depending on the satisfiability of formulas: the lack of Backjumping is less problematic with unsatisfiable formulas.

The second benchmark in run on hybrid formulas. Now we have 14 nominals and no propositional symbols, and the modal depth is 2. Clauses now look like:

```
N3 v [R1](-N6 v -[R1](N7 v -N1));
N4 v [R1](-N13 v [R1](-N6 v -N14));
-N14 v [R1](N12 v -[R1](N1 v N4));
-N9 v -[R1](-N1 v [R1](N6 v N4));
N2 v [R1](-N13 v -[R1](N3 v -N12));
```

We used batches of size $10, 30, \ldots 150$ with a timeout of 95 seconds. Figure 7.3 show the sat/unsat repartition of formulas, as assessed by HTab with all optimizations. Figure 7.4 show the median time results. We can see more distinction between optimizations in that case. Again, our implementation of Lazy Branching does not shine. Interestingly, Backjumping is not missed much, except in the area of maximum uncertainty, around size 50 and 70. Then, the absence of Eager Unit Propagation is more felt, and finally Semantic Branching and Unit Propagation both show that they bring a lot of improvement to the performance of HTab in that case.

Hybrid Provers comparison Now we compare HTab with Spartacus. The following benchmark involves formulas of the language $\mathcal{H}$ (i.e., no satisfaction operator) with 10 nominals, no propositional symbols, a maximal modal depth of 3. Clauses of the generated formulas look like:

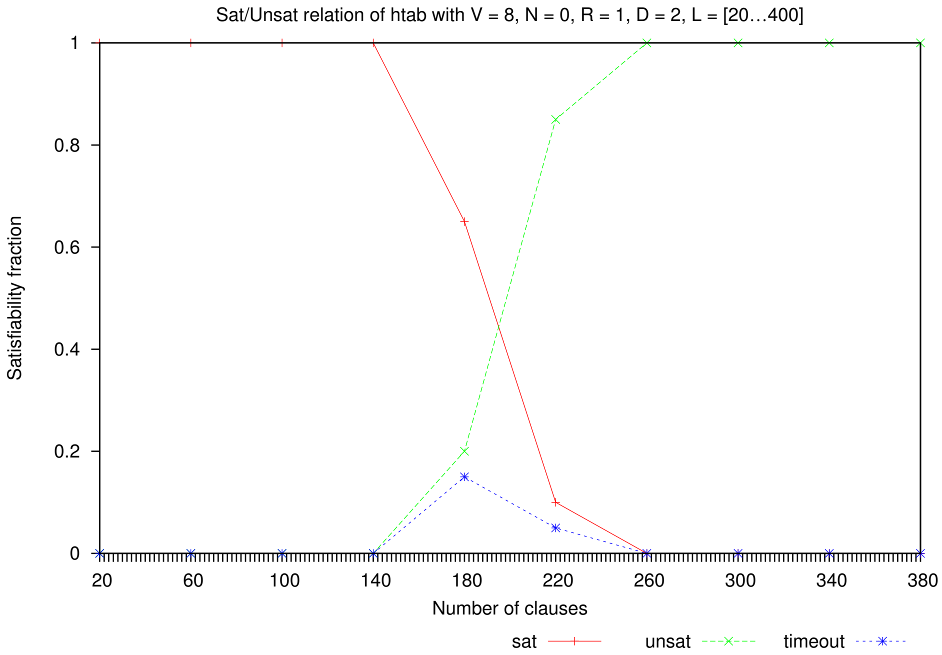


Figure 7.1: sat/unsat/timeout repartition of HTab with all optimizations

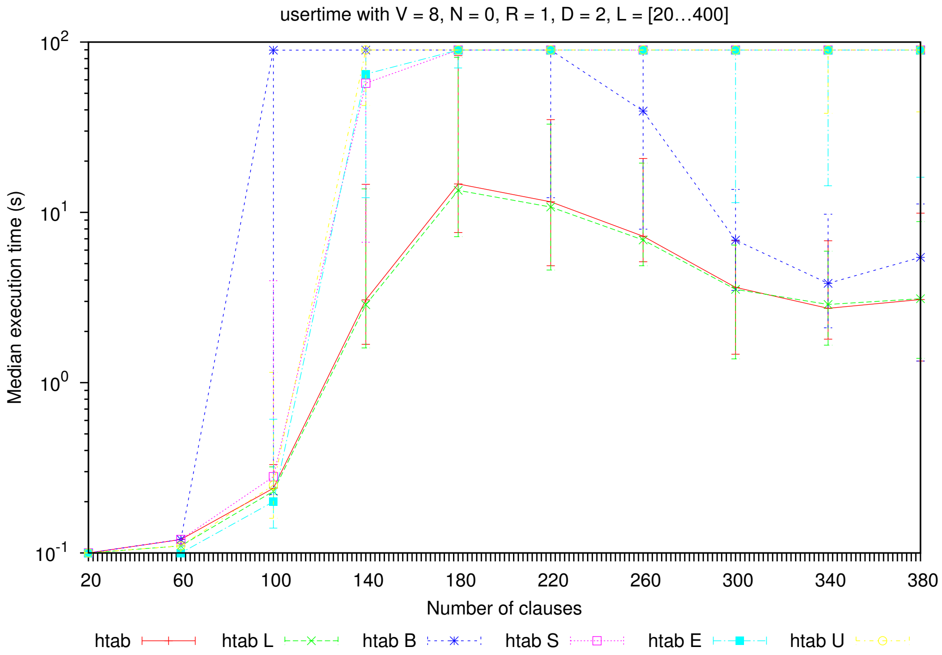


Figure 7.2: Median time of HTab versus HTab without each optimization

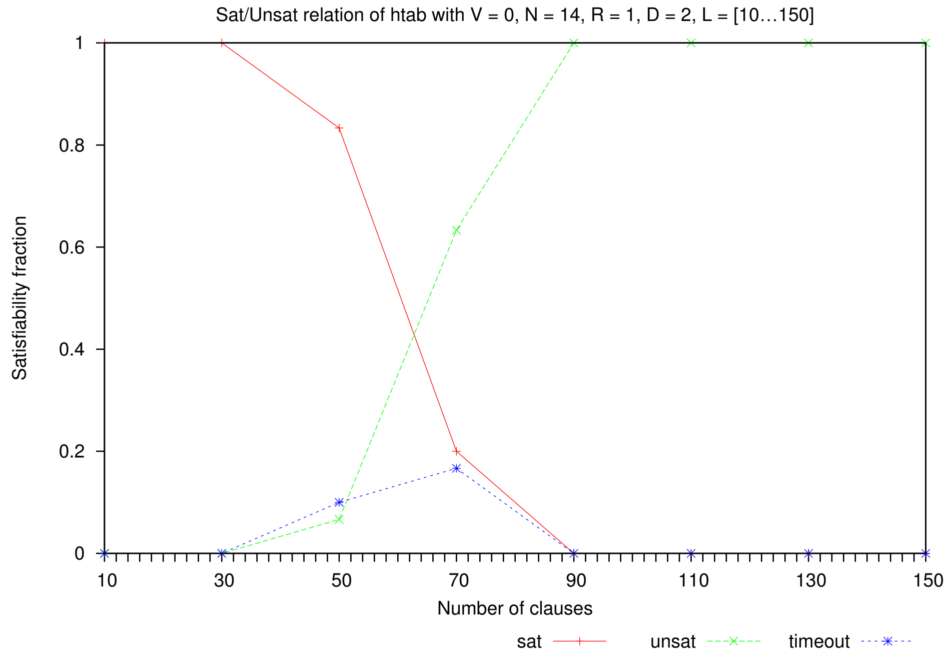


Figure 7.3: sat/unsat/timeout repartition of HTab with all optimizations

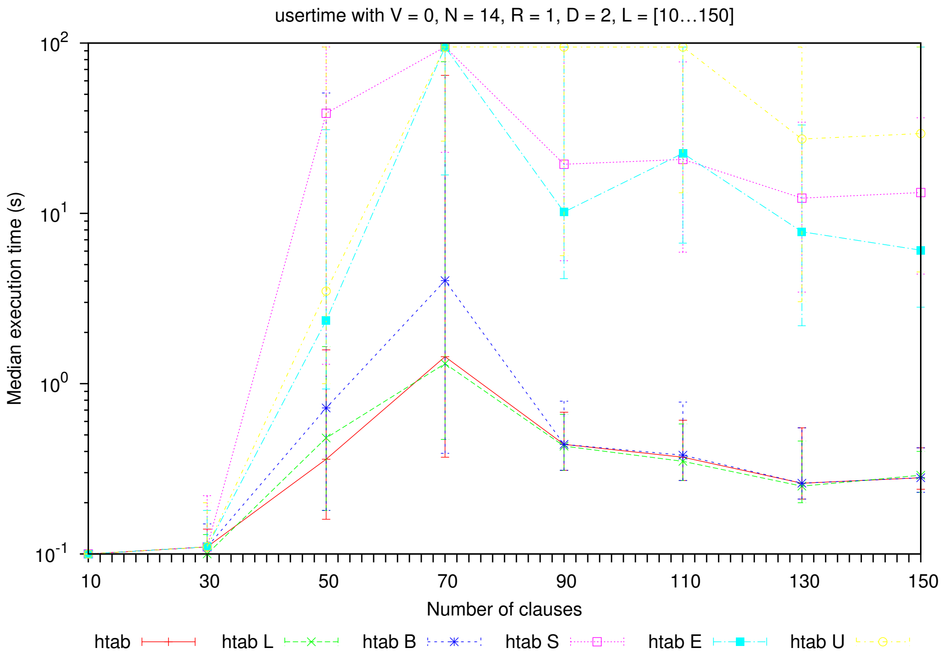


Figure 7.4: Median time of HTab versus HTab without each optimization

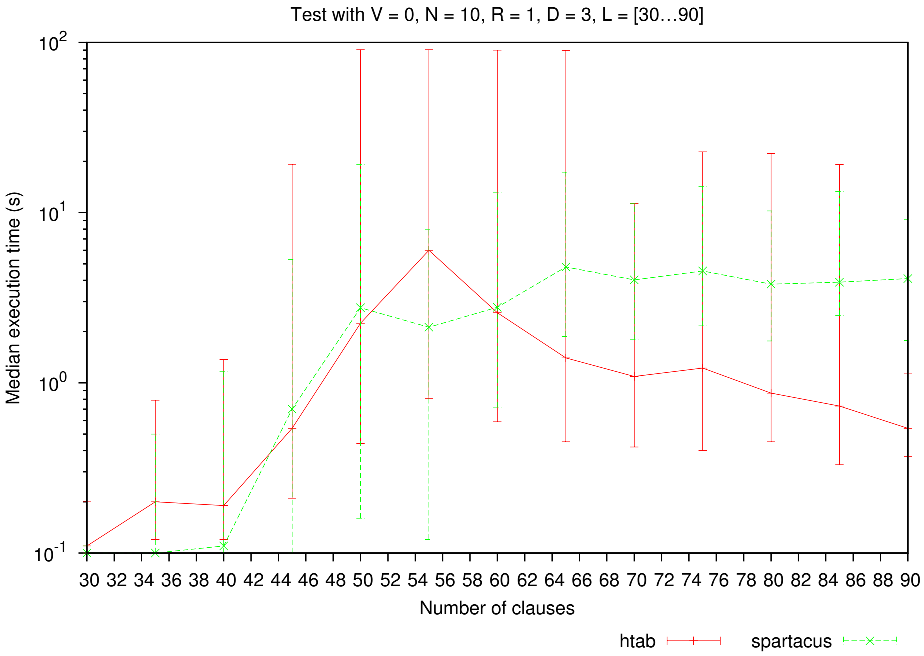


Figure 7.5: Median time of HTab and Spartacus

```
-N2 v -[R1](-N5 v [R1](N5 v -[R1](-N4 v -N1)));
-N9 v -[R1](N10 v [R1](-N3 v -[R1](P2 v -P1)));
-N3 v [R1](P2 v [R1](P5 v [R1](-N8 v -P3)));
-N3 v -[R1](N10 v [R1](N1 v -[R1](P4 v -P2)));
-N1 v -[R1](-N8 v -[R1](N9 v [R1](N9 v -P1)));
```

The batches we used were of size $30, 35, \ldots, 90$ and we used a timeout of 90 seconds. The median time graph obtained with one run of this benchmark is shown on Figure 7.5

We can make a few observations from this graph. HTab and Spartacus show comparable behaviours. While HTab struggles more on formulas of size between 30 and 60, it has a better median behaviour after this area, while Spartacus has a better median before 60 and a worse after.

However, let us have a look at the sat/unsat/timeout plots of HTab and Spartacus on Figures 7.6 and 7.7

By looking at the timeout lines, we can see that there are more cases where HTab does not answer within the given time than for Spartacus. Let us see the "unique answers" graph, which displays a count of how many times a given prover was the single one to give an answer, on Figure 7.8.

So, although the median time of Spartacus is worse, Spartacus can decide the satisfiability of more formulas than HTab. A conclusion to draw is that when running benchmarks we always need to look beyond the median time graph and take into account all available information: for practical purposes it may be better to wait

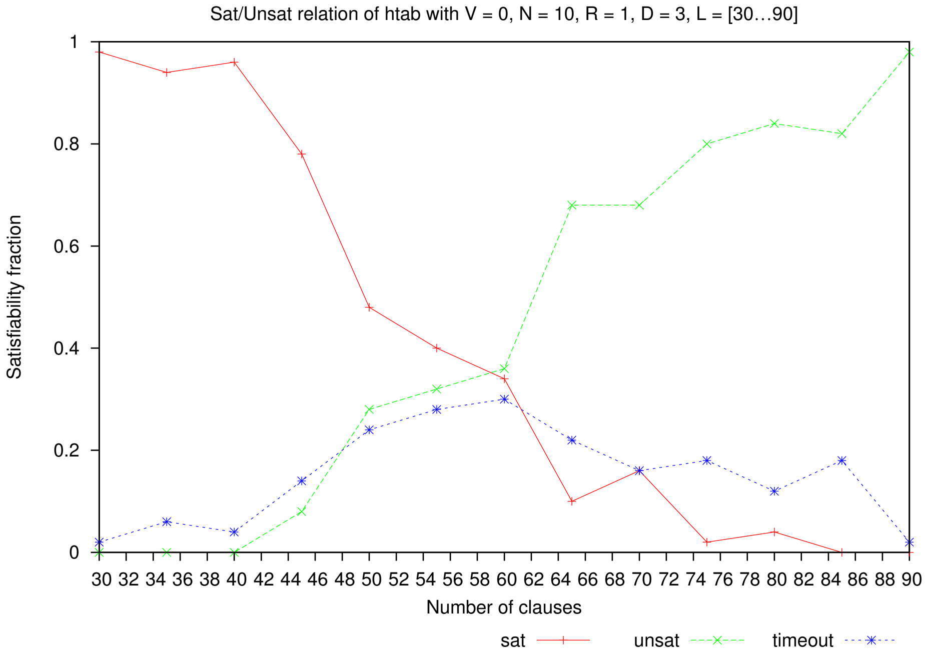


Figure 7.6: sat/unsat/timeout repartition of HTab

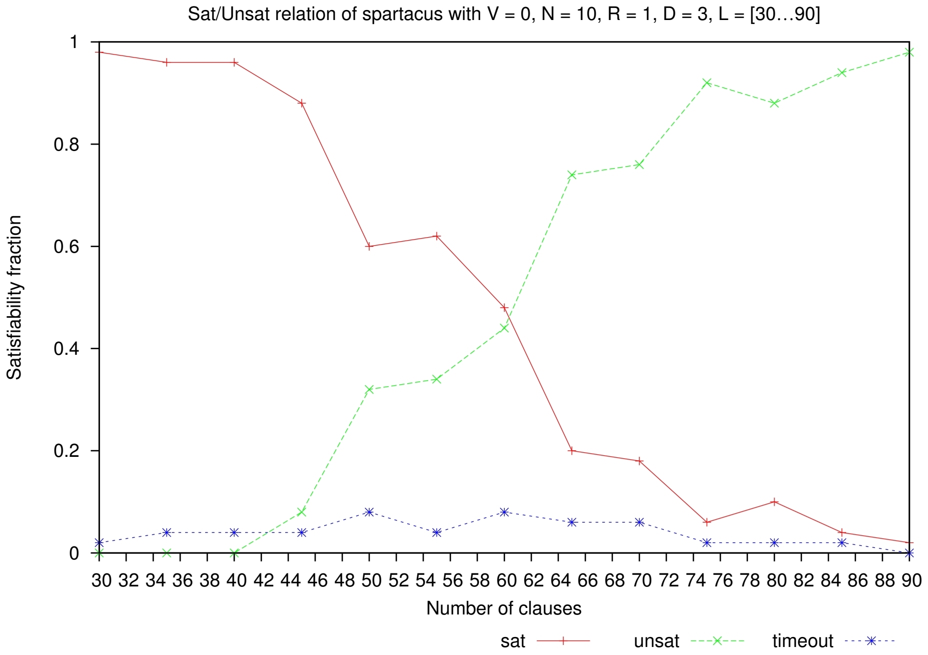


Figure 7.7: sat/unsat/timeout repartition of Spartacus

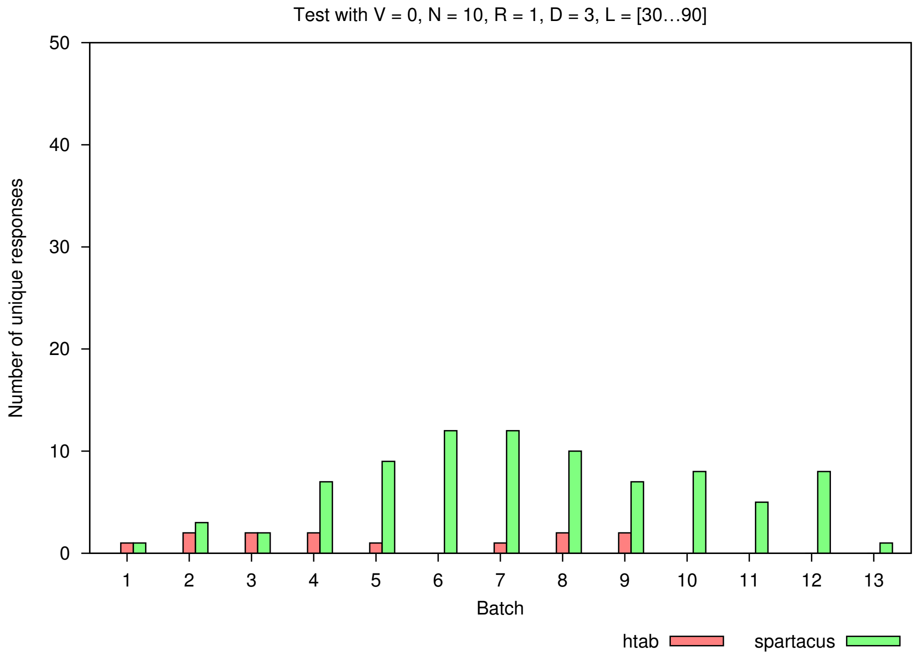


Figure 7.8: Unique responses of HTab and Spartacus

more time in general and have more answers.

We carried out a second benchmark with different parameters. These formulas contain 2 propositional symbols, 4 nominals, a modal depth of 3. Compared to other benchmarks, we augmented the number of disjuncts in every clause. Clauses now exhibit more diversity:

```
N4 v [R1](-P2 v -P1 v -[R1](-N3 v -P2 v -[R1](N2 v N3 v -P2)))
   v -[R1](N4 v -N1 v [R1](P2 v -N4 v P1));
-N1 v -P2
    v -[R1](-P1 v [R1](P2 v -P1 v -[R1](-N4 v P1 v -P2))
                v [R1](-P1 v P2 v -N2));
P1 v P2
   v [R1](N1 v P2 v [R1](N1 v -[R1](-P1 v P2 v N2)));
-N4 v -[R1](N1 v -N4 v -[R1](-N4 v -P1 v [R1](-P2 v -P1 v -N1)))
    v [R1](P2 v P1 v -[R1](N3 v P1 v -P2));
P1 v -N3
   v -[R1](-N2 v -N3 v -[R1](-P2 v [R1](-N1 v -N2 v -N4)));
```

Because of this greater number of disjuncts, we had to adjust the size of batches to bigger values in order to get a balanced sat-to-unsat repartition. Batches are of size 30, 40, . . . 300, and the timeout is 90 seconds. The sat/unsat/timeout repartition can be seen on Figure 7.9, the median times on Figure 7.10, and the unique answers on Figure 7.11.

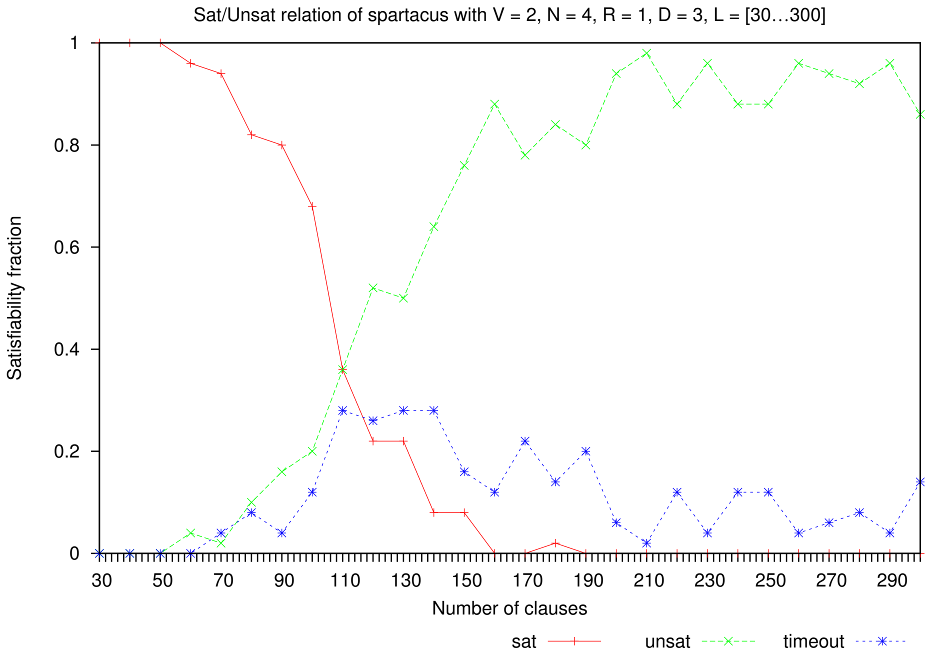


Figure 7.9: sat/unsat/timeout repartition for Spartacus

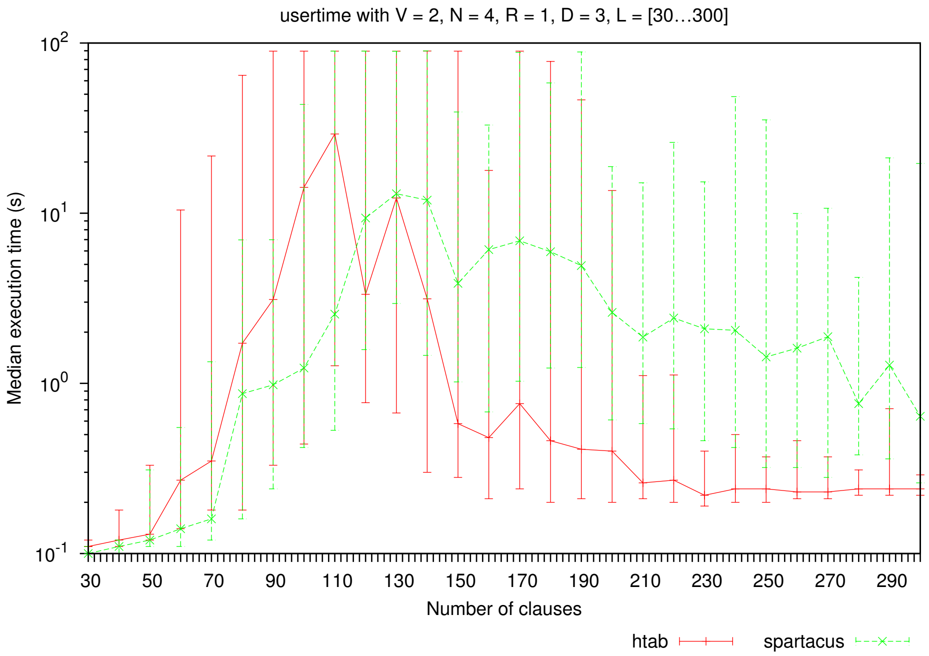


Figure 7.10: Median times of HTab and Spartacus

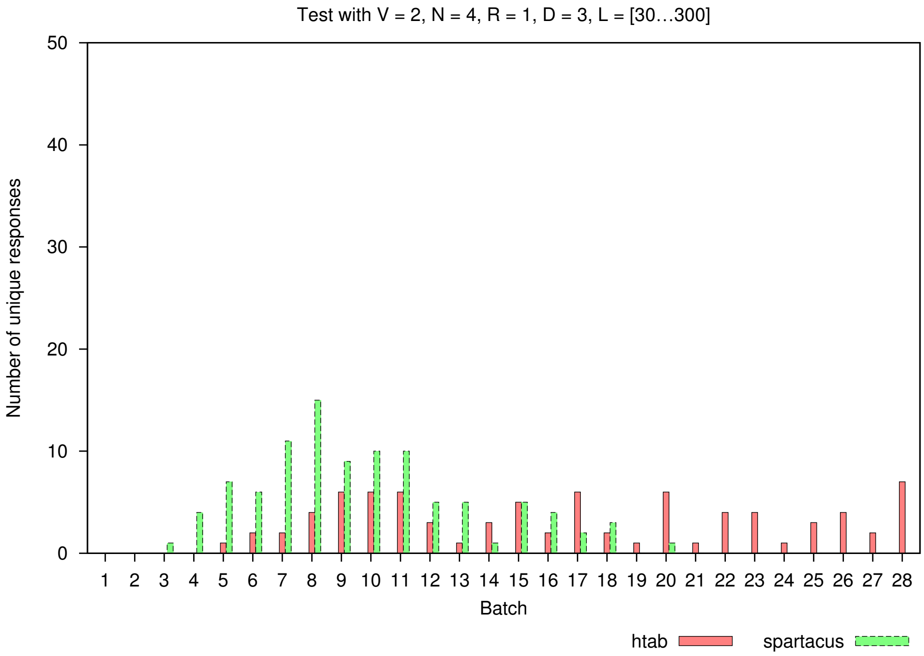


Figure 7.11: Unique responses of HTab and Spartacus

The Median Time and Unique Responses graphs enable us to see that Spartacus is more performant for satisfiable formulas, while HTab has the upper hand for unsatisfiable formulas.

Modal testing with description logic prover This time we include FaCT++ 1.5.0 (Tsarkov and Horrocks, 2006) in our tests. Due to what seems to be a bug in FaCT++ on some entries, we limited the timeout to 9 seconds to prevent crashes. We still maintained a sat-to-unsat repartition of formulas, but had to rely on easier formulas because of the low timeout value.

These provers run on the same modal formulas, except FaCT++, whose executable is called inside a wrapper that first converts the modal formulas into Description Logic TBoxes. Then FaCT++ checks satisfiability of the TBox it is given.

The modal formulas contain 8 propositional symbols, no nominals, and have a modal depth of 2. Clauses look like:

```
P7 v [R1](P4 v -[R1](P7 v -P4));
-P6 v [R1](-P1 v -[R1](P1 v P5));
P6 v -[R1](-P7 v -[R1](P3 v -P8));
-P4 v -[R1](-P3 v -[R1](-P8 v -P7));
P8 v -[R1](P7 v [R1](-P5 v -P3));
```

Moreover, translation of such a formula into a TBox description for FaCT++ looks like:

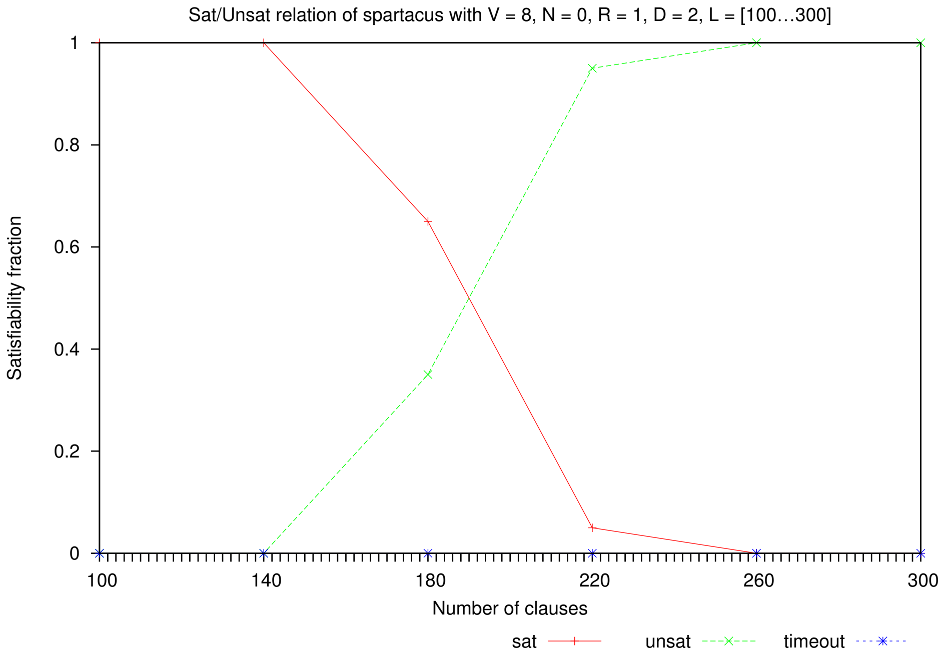


Figure 7.12: sat/unsat/timeout repartition for Spartacus

```
(defprimconcept P8)
(defprimconcept P7)
(defprimconcept P6)
(defprimconcept P5)
(defprimconcept P4)
(defprimconcept P3)
(defprimconcept P2)
(defprimconcept P1)
(defprimrole R1)
(defconcept Proof
 (and (or P6 (not (all R1 (or P5 (all R1 (or P2 P6))))))
  (and (or P7 (not (all R1 (or (not P2) (all R1 (or P1 P4))))))
   (and (or P5 (all R1 (or (not P4) (not (all R1 (or P8 P5))))))
    ...
```

We used batches of size 100,140,...,300, and the sat/unsat repartition can be seen on Figure 7.12. Median times are shown on Figure 7.13 and unique answers on Figure 7.14. The time taken by translating formulas for FaCT++ appears in the median time graph as the lowest, linear line. The result of this test is that both FaCT++ and HTab are quickly timeouting, while Spartacus can handle every formula of the test under the limit of 9 seconds.

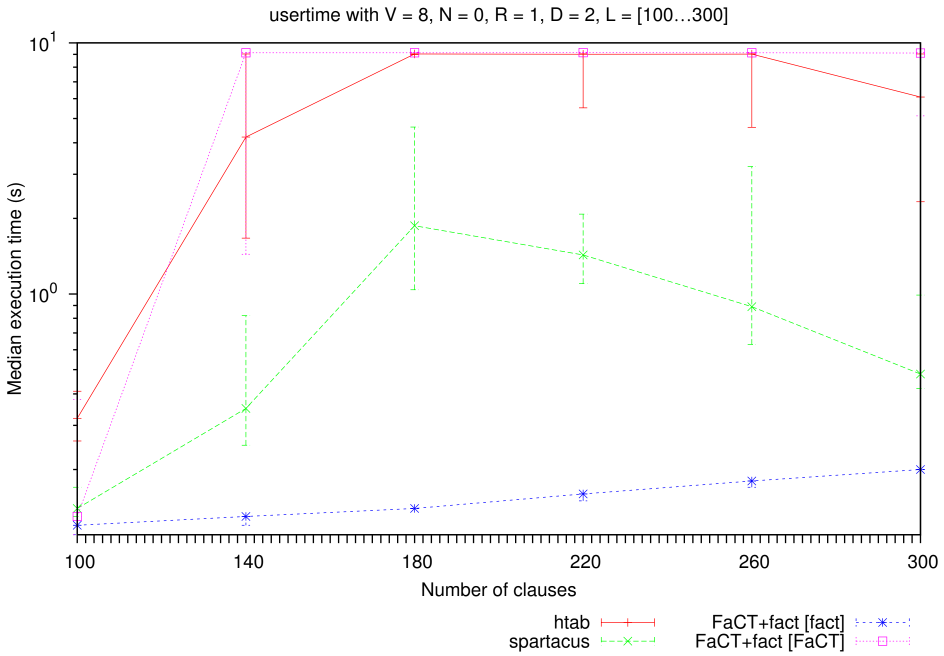


Figure 7.13: Median times of HTab, Spartacus and FaCT++

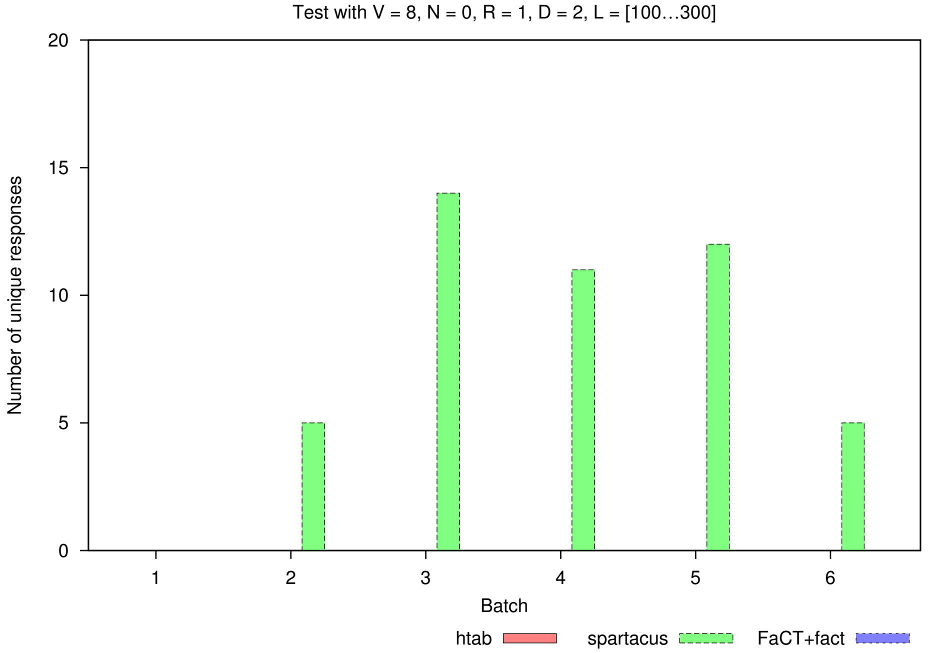


Figure 7.14: Unique responses among HTab, Spartacus and FaCT++

7.5 Perspectives

We now discuss some improvements to our set of testing tools that we would like to try out in the future.

Total time instead of median time

One easy change we would like to make to GridTest would be to let provers run as much time as needed on every formula of a benchmark, with only a global timeout limit for a whole batch of formulas. This would enable us to notice corner cases of bad performance, that currently do not appear in the median time graphs.

For instance, if a batch contains 100 formulas with a median time or 10 seconds, but has 10 formulas for which the prover take 1 minute to solve each, then the existence of these 10 formulas will never appear in the median time graph. One wants to know about these instances and why they give a harder time to the prover studied.

In that case, the benchmark report could include a plot for the total time spent on each batch, along with the number of formulas processed by batch. One potential problem we might face is that this number of processed formulas per batch may be non significative since they are not ordered by difficulty, contrary to some other benchmarks based on this total time limit (e.g, (Massacci, 1999; Heuerding and Schwendimann, 1996)).

Real-life problems

Random formulas are certainly the best thing when they are all that we have. But if we want to have good provers for real-life applications, we need them to be performant on real-life input. This input is different in the sense that it can contain patterns that can be exploited by provers (Nieuwenhuis et al., 2007). For example, the system described in (Gardiner et al., 2006a,b) can be used to collect real-life ontologies and test DL provers on them. We believe these kind of systems can also benefit from exploiting Grid computing: by assigning each example to a unique node the same linear speed-up we obtained should be expected.

Generating hard satisfiable problems

Another technique coming from the SAT domain is to look beyond purely random 3SAT formulas and try to generate random *hard* formulas. Aimed at incomplete reasoners that try to show satisfiability (as opposed to unsatisfiability) of formulas, some of these techniques work by hiding satisfying assignments (Achlioptas et al., 2005).

Expanding this technique to hybrid logic would be an interesting issue. One way to do it would be to generate models, and then to extract hard satisfiable hybrid formulas from it. Characterising a model with a modal formula is a known issue (Balbiani and Herzig, 2007). The challenge would be to generate *hard* characterizing formulas. For the initial step, we can build upon existing tools for generating random automata (Bassino et al., 2009).

This would give us a new source of hard satisfiable formulas, and we could require the provers to output models that satisfy the given formula, so that they would not cheat by systematically answering SAT.

This model-driven approach would provide an alternative to the current tools for random generation of modal and hybrid formulas that are mostly syntax-driven (Patel-Schneider and Sebastiani, 2003; Areces and Heguiabehere, 2003).

Chapter 8

Conclusions

This thesis is centered on reasoning tasks in various hybrid languages. We went from the presentation of tableaux-based deduction algorithms for hybrid logics, to their implementation and their evaluation. We also studied modal logics with counting, and saw their relations with hybrid logics. Let us summarize the main contributions of this thesis and draw some perspectives for future research.

Hybrid tableaux

We studied modal logic algorithms from the theoretical and implementational point of view. We presented a tableaux algorithm for the hybrid logic $\mathcal{H}(:, \mathsf{E}, \mathsf{D}, \Diamond^-)$. Termination of the calculus is ensured by using anywhere blocking for $\mathcal{H}(:, \mathsf{E}, \mathsf{D})$ and pairwise chain-based blocking for any language that involves the converse modality.

The aspect of model building was also considered: from an open branch, how does one build a model in which the input formula is true? Two approaches are possible: either filtrate the pre-model described by the branch, or avoid filtration and add the missing accessibility relations between nodes. The first approach is the one we used, the second is the one used by Kaminski and Smolka. Although the two approaches are clearly different, we feel that more investigation in their respective advantages and drawbacks would be valuable.

We presented extensions of the tableaux calculi for which we did not guarantee termination. For two of them, role hierarchies and functional and injective modalities, we believe we can obtain decision procedures under certain conditions. We have to be careful since previous research in description logics showed that combining role hierarchy, converse modalities and functional and injective modalities without any restriction lead to undecidability.

Counting

We studied the addition of counting operators to various logics. We noticed that, although this has been done for first-order, modal and description logics, the basic modal logic with counting operators, that we called $\mathcal{MLC}$, has never been studied as such.

We provide expressiveness results for this logic by means of a bisimulation. We also obtain complexity bounds for its satisfiability problem. These bounds are, partially, obtained from recent developments in complexity analysis of first-order logic with one and two variables with counting. We explicited the connexion between $\mathcal{MLC}$ and the hybrid logic $\mathcal{H}(:, \mathsf{E})$.

Finally we introduced a novel inference task which consists in finding, if it exists, the number of individuals in a model that satisfy a formula in the context of a theory. We provide an algorithm to carry out this task for any logic with counting operators, provided one has a decision procedure and model building procedure for it.

Since we have also studied decision and model building procedures for $\mathcal{H}(:, \mathsf{E})$ elsewhere in this thesis, we have at least the basic tools to carry out this task in modal logic with counting. However, the translation approach is very inefficient and could not be used for concrete implementations. Thus, future research on that question will involve finding and evaluating usable decision procedures and counting procedures.

Implementation

We showed an implementation of the calculus that we presented: the theorem prover HTab. We presented its architecture and optimizations. We also evaluated the impact of each optimization on its performance. HTab is now available, with its source code, to the community so as to encourage experimentation.

In the future, we would like to implement Pattern-Based Blocking into HTab and measure its performance. Having both techniques of anywhere blocking and pattern-based blocking in the same prover will provide useful information.

Another feature we would like to add to HTab would be to make it read the OWL format. Moreover, support for reading TPTP formulas could be also added, by using a translation from first-order logic to the hybrid logic $\mathcal{H}(:, \mathsf{E}, \downarrow)$. These two additions would open HTab to broader and harder problems, and would help us test it more intensively.

Methodology of evaluation

With regards to evaluation, we showed that we benefited greatly from our benchmarking suite GridTest. Since, to provide us with reliable results, benchmarks need to be run on a great number of formulas, they do take a lot of time if run on a single machine. The GridTest suite enables us to run a series of tests on several machines in parallel, decreasing the waiting time linearly in the number of machines.

However, there is much room for improvements. While keeping the basis of randomly generated formulas, we would like to have different kinds of tests at our disposal. Notably, the ability to evaluate the total time taken by a prover to run on a single batch of formulas, with a global timeout, would enable us to notice corner cases for which a prover would behave badly.

Bibliography

D. Achlioptas, H. Jia and C. Moore – "Hiding satisfying assignments: Two are better than one", *CoRR* **abs/cs/0503046** (2005). 138

M. Agrawal, N. Kayal and N. Saxena – "PRIMES is in P", *Annals of Mathematics* **160** (2004), no. 2, p. 781–793. 36

M. Alekhnovich, E. A. Hirsch and D. Itsykson – "Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas", *Journal of Automated Reasoning* **35** (2005), no. 1-3, p. 51–72. 37

I. H. Anellis – "From semantic tableaux to Smullyan trees: A history of the development of the falsifiability tree method.", *Mod. Log.* **1** (1990), no. 1, p. 36–69. 57

C. Areces – "Logic engineering. the case of description and hybrid logics", PhD Thesis, Institute for Logic, Language and Computation, University of Amsterdam, Amsterdam, The Netherlands, October 2000. 48

C. Areces and D. Gorín – "Coinductive models and normal forms for modal logics", *Logic Journal of the IGPL* (2010). 93

C. Areces and D. Gorín – "Ordered resolution with selection for h(@)", *Proceedings of LPAR 2004* (Montevideo, Uruguay) (F. Baader and A. Voronkov, eds.), LNCS, vol. 3452, Springer, 2005, p. 125–141. 124

C. Areces and J. Heguiabehere – "hGen: A random CNF formula generator for hybrid languages", *Methods for Modalities 3 - M4M-3, Nancy, France* (Nancy, France), September 2003. 18, 27, 125, 139

C. Areces and J. Heguiabehere – "Hylores: Direct resolution for hybrid logics", *Proceedings of Methods for Modalities 2* (Amsterdam, The Netherlands) (C. Areces and M. de Rijke, eds.), November 2001. 18, 107

C. Areces, P. Blackburn and M. Marx – "A road-map on complexity for hybrid logics", *Computer Science Logic* (Madrid, Spain) (J. Flum and M. Rodríguez-Artalejo, eds.), LNCS, no. 1683, Springer, 1999, Proceedings of the 8th Annual Conference of the EACSL, Madrid, September 1999., p. 307–321. 45

C. Areces, P. Blackburn and M. Marx – "The computational complexity of hybrid temporal logics", *Logic Journal of the IGPL* **8** (2000), no. 5, p. 653–679. 43, 103

C. Areces, P. Blackburn and M. Marx – "Hybrid logics: characterization, interpolation and complexity", *Journal of Symbolic Logic* **66** (2001), no. 3, p. 977–1010. 45

C. Areces, H. de Nivelle and M. de Rijke – "Resolution in modal, description and hybrid logic", *Journal of Logic and Computation* **11** (2001), no. 5, p. 717–736. 18

C. Areces, D. Gorín, A. Lorenzo and M. Pérez Rodríguez – "Testing provers on a grid - framework description", *Proceedings of the 22nd International Workshop on Description Logics (DL 2009)* (B. Cuenca Grau, I. Horrocks, B. Motik and U. Sattler, eds.), CEUR Workshop Proceedings, vol. 477, CEUR-WS.org, 2009. 18, 127

C. Areces, D. Gorín, A. Lorenzo and M. Perez Rodriguez – "Testing Provers on a Grid - Framework Description", *Proceedings of the 22nd International Workshop on Description Logics - DL 2009* (Oxford United Kingdom) (B. Cuenca Grau, I. Horrocks, B. Motik and U. Sattler, eds.), vol. 477, CEUR-WS.org, 2009. 124

S. Arora and B. Barak – *Computational complexity: A modern approach*, Cambridge University Press, New York, NY, USA, 2009. 31, 32

F. Baader and U. Sattler – "An overview of tableau algorithms for description logics", *Studia Logica* **69** (2000), p. 2001. 78

F. Baader, M. Buchheit and B. Hollunder – "Cardinality restrictions on concepts", *Artificial Intelligence* **88** (1996), no. 1–2, p. 195–213. 95

P. Balbiani and S. Demri – "Prefixed tableaux systems for modal logics with enriched languages", *Proceedings of the fifteenth international joint conference on artificial intelligence, IJCAI-97* (M. E. Pollack, ed.), vol. I, 1997, p. 190–195. 70

P. Balbiani and A. Herzig – "Talkin'bout Kripke models", *HyLo workshop 2007*, 2007. 138

F. Bassino, J. David and C. Nicaud – "Enumeration and random generation of possibly incomplete deterministic automata", *Pure Mathematics and Applications* (2009). 139

E. W. Beth – "Semantic entailment and formal derivability", *Koninklijke Nederlandse Akademie van Wentenschappen, Proceedings of the Section of Sciences* **18** (1955), p. 309–342. 56

T. Bittner and M. Donnelly – "Logical properties of foundational relations in bio-ontologies", *Artificial Intelligence in Medicine* **39** (2007), no. 3, p. 197–216. 94

P. Blackburn – "Internalizing labelled deduction", *Journal of Logic and Computation* **10** (2000), no. 1, p. 137–168. 55, 57, 60

P. Blackburn – "Nominal tense logic", *Notre Dame Journal of Formal Logic* **34** (1993), p. 56–83. 43

P. Blackburn and J. Seligman – "Hybrid languages", *Journal of Logic, Language and Information* **4** (1995), p. 251–272. 45

P. Blackburn, M. de Rijke and Y. Venema – *Modal logic*, Cambridge University Press, 2001. 92, 101, 102

P. Blackburn and J. van Benthem – "Modal logic: A semantic perspective", Handbook of Modal Logic, Elsevier North-Holland, 2006. 50

P. Blackburn, M. de Rijke and Y. Venema – *Modal logic*, Cambridge Tracts in Theoretical Computer Scie, vol. 53, Cambridge University Press, Cambridge, 2001. 42

T. Bolander and P. Blackburn – "Termination for hybrid tableaus", *Journal of Logic and Computation* **17** (2007), no. 3, p. 517–554. 12, 26, 55, 57, 59, 69, 80

T. Bolander and P. Blackburn – "Terminating tableau calculi for hybrid logics extending K", *Methods for Modalities 5* (Cachan, France) (S. Demri and C. Areces, eds.), 2007. 58, 87

T. Bolander and T. Braüner – "Tableau-based Decision Procedures for Hybrid Logic", *Journal of Logic and Computation* **16** (2006), no. 6, p. 737–763. 57

R. Brummayer and A. Biere – "Fuzzing and delta-debugging SMT solvers", *SMT '09: Proceedings of the 7th International Workshop on Satisfiability Modulo Theories* (New York, NY, USA), ACM, 2009, p. 1–5. 20, 27

S. Cerrito and M. C. Mayer – "Terminating tableaux for HL(@) without loop-checking", Tech. Report IBISC-RR-2007-07, Ibisc Lab., Université d'Evry Val d'Essonne, 2007. 58

S. Cerrito and M. C. Mayer – "An efficient approach to nominal equalities in hybrid logic tableaux", *Journal of Applied Non-Classical Logics* **20** (2010). 58

A. Cobham – "The intrinsic computational difficulty of functions", *Logic, Methodology and Philosophy of Science, proceedings of the second International Congress, held in Jerusalem, 1964* (Amsterdam) (Y. Bar-Hillel, ed.), North-Holland, 1965. 34

S. A. Cook – "The complexity of theorem-proving procedures", *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing* (New York, NY, USA), ACM, 1971, p. 151–158. 37

T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein – *Introduction to algorithms*, The MIT Press, 2001. 14, 69

M. Davis and H. Putnam – "A computing procedure for quantification theory", *J. ACM* **7** (1960), no. 3, p. 201–215. 36

M. Davis, G. Logemann and D. Loveland – "A machine program for theorem-proving", *Commun. ACM* **5** (1962), no. 7, p. 394–397. 36

G. De Giacomo and M. Lenzerini – "Description logics with inverse roles, functional restrictions, and n-ary relations", *In Proc. of JELIA-94, volume 838 of LNAI*, Springer-Verlag, 1994, p. 332–346. 86

G. De Giacomo and F. Massacci – "Combining deduction and model checking into tableaux and algorithms for converse-PDL", *Information and Computation* **162** (2000), no. 1/2, p. 117–137. 114

M. de Rijke – "The modal logic of inequality", *Journal of Symbolic Logic* **57** (1992), no. 2, p. 566–584. 42

M. De Rijke – "A note on graded modal logic", *Studia Logica* **64** (2000), p. 271–283. 92

S. Demri – "A simple tableau system for the logic of elsewhere", *Lecture Notes in Computer Science* **1071** (1996), p. 177–192. 42, 70

J. Edmonds – "Paths, trees, and flowers", *Canad. J. Math.* **17** (1965), p. 449–467. 34

J. v. Eijck – "HyLoTab — Tableau-based theorem proving for hybrid logics", manuscript, CWI, available from `http://www.cwi.nl/~jve/hylotab`, 2002. 18

J. Faddoul, N. Farsinia, V. Haarslev and R. Möller – "A hybrid tableau algorithm for ALCQ", *Proc. of ECAI 2008* (Amsterdam, The Netherlands), IOS Press, 2008, p. 725–726. 105

L. Fariñas Del Cerro, D. Fauthoux, O. Gasquet, A. Herzig and D. Longin – "LoTREC: the generic tableau prover for modal and description logics", *In International Joint Conference on Automated Reasoning, LNCS*, Springer Verlag, 2001, p. 453–458. 119

S. Figueira and D.Gorín – "On the size of shortest modal descriptions", *Proceedings of AIML 2010*, 2010. 56

K. Fine – "In so many possible worlds", *Notre Dame Journal of Formal Logics* **13** (1972), no. 4, p. 516–520. 16, 91, 92

M. Fitting – "Tableau methods of proof for modal logics", *Notre Dame Journal of Formal Logic* **13** (1972), no. 2, p. 237–247. 57

T. Gardiner, I. Horrocks and D. Tsarkov – "Automated benchmarking of description logic reasoners", *Proceedings of the 2006 Description Logic Workshop (DL 2006)*, CEUR (`http://ceur-ws.org/`), vol. 189, 2006. 123, 138

T. Gardiner, D. Tsarkov and I. Horrocks – "Framework for an automated comparison of description logic reasoners", *Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, Lecture Notes in Computer Science, vol. 4273, Springer, 2006, p. 654–667. 123, 138

G. Gargov and V. Goranko – "Modal logic with names", *Journal of Philosophical Logic* **22** (1993), no. 6, p. 607–636. 42, 43, 44

O. Gasquet, A. Herzig, D. Longin and M. Sahade – "LoTREC: Logical tableaux research engineering companion", *In TABLEAUX*, Springer, 2005, p. 318–322. 119

I. P. Gent and T. Walsh – "The SAT phase transition", *Proc. of the 11th European Conference on Artificial Intelligence* (Amsterdam, The Netherlands) (A. Cohn, ed.), 1994, p. 105–109. 124

I. P. Gent and T. Walsh – "Beyond NP: the QSAT phase transition", AAAI Press, 1999, p. 648–653. 124

G. Gentzen – "Untersuchungen über das logische schließen. ii", *Mathematische Zeitschrift* **39** (1935), p. 405–431, 10.1007/BF01201363. 56

E. Giunchiglia, F. Giunchiglia, R. Sebastiani and A. Tacchella – "More evaluation of decision procedures for modal logics", KR'98: Principles of Knowledge Representation and Reasoning, Morgan Kaufmann, 1998, p. 626–635. 123

F. Giunchiglia and R. Sebastiani – "A sat-based decision procedure for alc", *In Proc. of the 5th International Conference on Principles of Knowledge Representation and Reasoning - KR'96*, Morgan Kaufmann, 1996, p. 304–314. 123

D. Goldin and P. Wegner – "The Church-Turing thesis: Breaking the myth", *New Computational Paradigms* (2005), p. 152–168. 33

V. Goranko – "Temporal logic with reference pointers", Temporal logic, LNCS, vol. 827, Berlin: Springer, 1994, p. 133–148. 44

V. Goranko and S. Passy – "Using the universal modality: Gains and questions", *Journal of Logic and Computation* 2 (1992), no. 1, p. 5–30. 41

R. Goré and L. A. Nguyen – "Exptime tableaux with global caching for description logics with transitive roles, inverse roles and role hierarchies", *TABLEAUX '07: Proceedings of the 16th international conference on Automated Reasoning with Analytic Tableaux and Related Methods*, Springer-Verlag, 2007, p. 133–148. 56

R. Goré and L. Postniece – "An experimental evaluation of global caching for ALC (system description)", *IJCAR '08: Proceedings of the 4th international joint conference on Automated Reasoning*, Springer-Verlag, 2008, p. 299–305. 56

R. Goré and F. Widmann – "Optimal and cut-free tableaux for propositional dynamic logic with converse", *IJCAR* (J. Giesl and R. Hähnle, eds.), Lecture Notes in Computer Science, vol. 6173, Springer, 2010, p. 225–239. 114

D. Gorín – "Automated reasoning techniques for hybrid logics", PhD Thesis, Universidad de Buenos Aires Facultad de Ciencias Exactas y Naturales Departamento de Computación, UFR STMIA - Ecole doctorale IAEM Lorraine Département de formation doctorale en informatique, 2009. 18, 107

D. Götzmann, M. Kaminski and G. Smolka – "Spartacus: A tableau prover for hybrid logic", *M4M-6* (T. Bolander and T. Braüner, eds.), ENTCS, vol. 262, Roskilde University, Elsevier, May 2010, p. 127–139. 17, 18, 111, 117, 119

E. Grädel, M. Otto and E. Rosen – "Two-variable logic with counting is decidable", *Proceedings of LICS 97*, 1997, p. 306–317. 91

E. Grädel, P. G. Kolaitis and M. Y. Vardi – "On the decision problem for two-variable first-order logic", *Bulletin of Symbolic Logic* **3** (1997), no. 1, p. 53–69. 91

V. Haarslev and R. Möller – "Description of the Racer System and its Applications", *Proceedings of the 2001 International Description Logics Workshop* (C. Goble, D. L. McGuinness, R. Möller and P. F. Patel-Schneider, eds.), 2001, p. 46–55. 94

V. Haarslev and R. Möller – "On the scalability of description logic instance retrieval", *Journal of Automated Reasoning* **41** (2008), p. 99–142, 10.1007/s10817-008-9104-7. 103

V. Haarslev, M. Timmann and R. Möller – "Combining tableaux and algebraic methods for reasoning with qualified number restrictions", *Proc. of Description Logics 2001*, 2001, p. 152–161. 105

J. Halpern and Y. Moses – "A guide to completeness and complexity for modal logics of knowledge and belief", *Artificial Intelligence* **54** (1992), p. 319–379. 39, 70, 71

A. Heuerding and S. Schwendimann – "A benchmark method for the propositional modal logics K, KT, and S4", Technical report IAM-96-015, University of Bern, Switzerland, 1996. 123, 138

J. Hintikka – "Form and content in quantification theory", *Acta Philosophica Fennica — Two papers on Symbolic Logic* **8** (1955), p. 8–55. 56

J. Hintikka – "Notes on quantification theory", *Societas Scientificarum Fennica, Commentationes Physico-Mathematicae* **17** (1955). 56

G. Hoffmann and C. Areces – "HTab: a terminating tableaux system for hybrid logic", *Electron. Notes Theor. Comput. Sci.* **231** (2009), p. 3–19, Also in Proc. of Method For Modalities 5, 2007. 17, 18, 27, 119

I. Horrocks and U. Sattler – "A description logic with transitive and inverse roles and role hierarchies", *Journal of Logic and Computation* **9** (1999), no. 3, p. 385–410. 84

I. Horrocks, U. Sattler and S. Tobies – "Practical reasoning for expressive description logics", *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)* (H. Ganzinger, D. McAllester and A. Voronkov, eds.), Lecture Notes in Artificial Intelligence, no. 1705, Springer, 1999, p. 161–180. 94, 113

I. Horrocks, P. Patel-Schneider and R. Sebastiani – "An analysis of empirical testing for modal decision procedures", *Logic Journal of the IGPL* **8** (2000), p. 293–323. 123

I. Horrocks – "Ontologies and the semantic web", *Communications of the ACM* **51** (2008), no. 12, p. 58–67. 6, 25

I. Horrocks and P. F. Patel-Schneider – "Optimizing description logic subsumption", *J. of Logic and Computation* **9** (1999), no. 3, p. 267–293. 115, 116

I. Horrocks and U. Sattler – "A tableau decision procedure for SHOIQ", *Journal of Automated Reasoning* **39** (2007), no. 3, p. 249–276. 94

I. Horrocks, U. Sattler and S. Tobies – "Practical reasoning for description logics with functional restrictions", *Proceedings of the 1999 Workshop Methods for Modalities (M4M-1)*, 1999. 86

U. Hustadt and R. Schmidt – "On evaluating decision procedures for modal logic", *Proc. of the 15th IJCAI* (M. Pollack, ed.), 1997, p. 202–207. 123

U. Hustadt and R. A. Schmidt – "A comparison of solvers for propositional dynamic logic", *Proceedings of the Second International Workshop on Practical Aspects of Automated Reasoning (PAAR-2010), Edinburgh, UK, July 14, 2010* (B. Konev, R. Schmidt and S. Schulz, eds.), 2010, p. 60–69. 115

M. Kaminski, S. Schneider and G. Smolka – "Terminating tableaux for graded hybrid logic with global modalities and role hierarchies", *TABLEAUX 2009* (M. Giese and A. Waaler, eds.), LNCS (LNAI), vol. 5607, Springer, 2009, p. 235–249. 58

M. Kaminski and G. Smolka – "Terminating tableau systems for hybrid logic with difference and converse", *Journal of Logic, Language and Information* **18** (2009), no. 4, p. 437–464. 58, 59, 69, 80

M. Kaminski and G. Smolka – "A straightforward saturation-based decision procedure for hybrid logic", *International Workshop on Hybrid Logic 2007 (HyLo 2007)* (Dublin, Ireland) (J. Villadsen, T. Bolander and T. Braüner, eds.), Aug 2007. 58, 77

M. Kaminski and G. Smolka – "Hybrid tableaux for the difference modality", *Electronic Notes in Theoretical Computer Science* **231** (2009), p. 241–257. 14, 18, 26, 58, 59, 69, 70, 100, 105

M. Kaminski and G. Smolka – "Clausal graph tableaux for hybrid logic with eventualities and difference", *LPAR-17* (C. Fermüller and A. Voronkov, eds.), LNCS (ARCoSS), vol. 6397, Springer, Oct 2010, p. 417–431. 58, 114

M. Kaminski and G. Smolka – "Terminating tableaux for hybrid logic with eventualities", *IJCAR 2010*, LNCS, Springer, Jul 2010. 58, 114

M. Kaminski and G. Smolka – "Terminating tableaux for SOQ with number restrictions on transitive roles", *TCS 2010* (C. S. Calude and V. Sassone, eds.), IFIP AICT, vol. 323, Springer, Sep 2010, p. 213–228. 58, 91, 92, 93, 98

M. Kaminski, S. Schneider and G. Smolka – "Terminating tableaux for graded hybrid logic with global modalities and role hierarchies", *TABLEAUX 2009* (M. Giese and A. Waaler, eds.), LNCS (LNAI), vol. 5607, Springer, Jul 2009, p. 235–249. 91, 93, 95, 104

Y. Kazakov and I. Pratt-Hartmann – "A note on the complexity of the satisfiability problem for graded modal logics", *LICS '09: Proceedings of the 2009 24th Annual IEEE Symposium on Logic In Computer Science* (Washington, DC, USA), IEEE Computer Society, 2009, p. 407–416. 92, 94, 103

Y. Kazakov, U. Sattler and E. Zolin – "How many legs do I have? non-simple roles in number restrictions revisited", *LPAR*, Lecture Notes in Computer Science, vol. 4790, Springer, 2007, p. 303–317. 94

S. Kripke – "A Completeness Theorem in Modal Logic", *J. Symb. Log.* **24** (1959), no. 1, p. 1–14. 38

S. Kripke – "Semantical considerations on modal logic", *Acta Philos. Fennica* **16** (1963), p. 83–94. 38

R. Ladner – "The computational complexity of provability in systems of modal logic", *SIAM Journal on Computing* **6** (1977), p. 467–480. 41

M. Marx – "Narcissists, stepmothers and spies", *In Proc. of DL'02, volume 53. CEUR*, 2002. 48

F. Massacci – "Design and results of the Tableaux-99 non-classical (modal) system competition", *Proc. Tableaux'99*, 1999. 123, 138

M. C. Mayer and S. Cerrito – "Herod and pilate: two tableau provers for basic hybrid logic", *IJCAR 2010, LNAI 6173* (J. Giesl and R. Haehnle, eds.), 2010, p. 255–262. 119

A. Mostowski – "On a generalization of quantifiers", *Fundamenta Mathematicae* **44** (1957), p. 12–36. 89, 90

B. Motik, R. Shearer and I. Horrocks – "Optimized reasoning in description logics using hypertableaux", *Proc. of the 21st Int. Conf. on Automated Deduction (CADE-21)*, Lecture Notes in Artificial Intelligence, vol. 4603, Springer, 2007, p. 67–83. 94

T. Mytkowicz, A. Diwan, M. Hauswirth and P. F. Sweeney – "Producing wrong data without doing anything obviously wrong!", *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems* (New York, NY, USA), ACM, 2009, p. 265–276. 126

R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell and A. Rubio – "Challenges in satisfiability modulo theories", *RTA'07: Proceedings of the 18th international conference on Term rewriting and applications* (Berlin, Heidelberg), Springer-Verlag, 2007, p. 2–18. 138

E. Nuutila – *Efficient transitive closure computation in large digraphs*, Mathematics and Computing in Engineering Series, vol. 74, Finnish Academy of Technology, 1995. 76

H. J. Ohlbach and J. Koehler – "Modal logics, description logics and arithmetic reasoning", *Artif. Intell.* **109** (1999), no. 1-2, p. 1–31. 105

L. Pacholski, W. Szwast and L. Tendera – "Complexity of two-variables logic with counting", *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science (LICS'97)*, IEEE Computer Society Press, 1997, p. 318–327. 91, 95

L. Pacholski, W. Szwast and L. Tendera – "Complexity of two-variables logic with counting", *SIAM Journal on Computing* **29** (2000), no. 4, p. 1083–1117. 91

E. Pacuit and S. Salame – "Majority logic", *In KR 2004, Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference*, AAAI Press, 2004, p. 598–605. 89, 91

S. Passy and T. Tinchev – "Quantifiers in combinatory PDL: completeness, definability, incompleteness", Fundamentals of Computation Theory FCT 85, LNCS, vol. 199, Springer, 1985, p. 512–519. 43

S. Passy and T. Tinchev – "PDL with data constants", *Information Processing Letters* **20** (1985), p. 35–41. 43

P. Patel-Schneider and R. Sebastiani – "A new general method to generate random modal formulae for testing decision procedures", *Journal of Artificial Intelligence Research* **18** (2003), p. 351–389. 123, 124, 125, 139

P. F. Patel-Schneider – "DLP system description", *Collected Papers from the International Description Logics Workshop (DL'98)*, 1998, p. 87–89. 56

V. Pratt – "Every prime has a succinct certificate", *SIAM Journal of Computing* 4 (1975), p. 214–220. 36

I. Pratt-Hartmann – "Complexity of the two-variable fragment with counting quantifiers", *Journal of Logic, Language and Information* **14** (2005), p. 369–395, 10.1007/s10849-005-5791-1. 26, 91, 95, 103

I. Pratt-Hartmann – "On the computational complexity of the numerically definite syllogistic and related logics", *Bulletin of Symbolic Logic* **14** (2008), no. 1, p. 1–28. 26, 91, 93, 103

I. Pratt-Hartmann – "The two-variable fragment with counting revisited", Logic, Language, Information and Computation (A. Dawar and R. de Queiroz, eds.), Lecture Notes in Computer Science, vol. 6188, Springer Berlin / Heidelberg, 2010, p. 42–54. 91

A. Prior – *Past, present and future*, Oxford University Press, 1967. 43

A. Prior – "Now", *Nous* **2** (1968), p. 101–119. 43

P. Pudlák and R. Impagliazzo – "A lower bound for DLL algorithms for k-SAT", *Symposium on Discrete Algorithms*, 2000, p. 128–136. 37

A. Rector and I. Horrocks – "Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions", *Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97)*, AAAI Press, Menlo Park, California, 1997. 94

J. B. Rosser – "An Informal Exposition of Proofs of Gödel's Theorems and Church's Theorem", *Journal of Symbolic Logic* **4** (1939), no. 2, p. 53–60. 33

B. Said – "Réécriture de graphes pour la construction de modèles en logique modale", PhD Thesis, Université de Toulouse, Toulouse, France, janvier 2010. 119

U. Sattler – "A concept language extended with different kinds of transitive roles", *KI* (G. Görz and S. Hölldobler, eds.), Lecture Notes in Computer Science, vol. 1137, Springer, 1996, p. 333–345. 114

W. J. Savitch – "Relationships between nondeterministic and deterministic tape complexities", *Journal of Computer and System Sciences* **4** (1970), no. 2, p. 177 – 192. 35

A. Schaerf – "Reasoning with individuals in concept languages", *Data Knowledge Engineering* **13** (1994), no. 2, p. 141–176. 47

M. Schmidt-Schauss and G. Smolka – "Attributive concept descriptions with complements", *Artificial Intelligence* **48** (1991), p. 1–26. 57

T. Schneider – "The complexity of hybrid logics over restricted frame classes", PhD Thesis, University of Jena, 2007. 45, 70

L. Schröder and D. Pattinson – "How many toes do I have? parthood and number restrictions in description logics", *Proc. Knowledge Representation 2008*, AAAI Press, 2008. 94

L. Schröder and D. Pattinson – "How many toes do I have? parthood and number restrictions in description logics", *Proc. Knowledge Representation 2008*, AAAI Press, 2008. 94

E. Sirin, B. C. Grau and B. Parsia – "Optimizing description logic reasoning with nominals: First results", Tech. report, 2004. 97

E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur and Y. Katz – "Pellet: A practical OWL-DL reasoner", *Web Semantics: Science, Services and Agents on the World Wide Web* **5** (2007), no. 2, p. 51 – 53, Software Engineering and the Semantic Web. 94

R. M. Smullyan – *First-order logic*, second corrected ed., Dover Publications, New York, 1995, First published in 1968 by. 56

E. Spaan – "Complexity of modal logics", PhD Thesis, 1993. 42

M. Suda, C. Weidenbach and P. Wischnewski – "On the saturation of YAGO", *Automated Reasoning* (J. Giesl and R. Hähnle, eds.), Lecture Notes in Computer Science, vol. 6173, Springer Berlin, Heidelberg, 2010, p. 441–456. 6, 25

G. Sutcliffe, C. Suttner and T. Yemenis – "The TPTP problem library", *Proc. of CADE-12* (Nancy, France) (A. Bundy, ed.), LNAI, no. 814, 1994, p. 252–266. 122

B. ten Cate – "Model theory for extended modal languages", PhD Thesis, University of Amsterdam, 2005, ILLC Dissertation Series DS-2005-01. 45

B. ten Cate and M. Franceschet – "On the complexity of hybrid logics with binders", *Proceedings of Computer Science Logic 2005* (L. Ong, ed.), Lecture Notes in Computer Science, vol. 3634, Springer Verlag, 2005, p. 339–354. 45, 50, 112

The OAR team – "The OAR resource manager", http://oar.imag.fr, 2010. 127

S. Tobies – "PSPACE reasoning for DLs with qualifying number restrictions", Tech. report, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 1999, `http://www-lti.informatik.rwth-aachen.de/Forschung/Papers.html`. 94

S. Tobies – "Complexity results and practical algorithms for logics in knowledge representation", PhD Thesis, LuFG Theoretical Computer Science, RWTH-Aachen, 2001. 48, 95, 103

S. Tobies – "PSPACE reasoning for graded modal logics", *Journal of Logic and Computation* **11** (2001), no. 1, p. 85–106. 92

D. Tsarkov, I. Horrocks and P. F. Patel-Schneider – "Optimizing terminological reasoning for expressive description logics", *Journal of Automated Reasoning* **39** (2007), no. 3, p. 277–316. 84

D. Tsarkov and I. Horrocks – "FaCT++ description logic reasoner: System description", *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, Lecture Notes in Artificial Intelligence, vol. 4130, Springer, 2006, p. 292–297. 135

A. M. Turing – "On computable numbers, with an application to the Entscheidungsproblem", *Proceedings of the London Mathematical Society* **2** (1936), no. 42, p. 230–265. 31

M. Tzakova – "Tableau calculi for hybrid logics", *Proceedings of the Conference on Tableaux Calculi and Related Methods (TABLEAUX)* (Saratoga Springs, USA) (N. Murray, ed.), vol. 1617, Springer Verlag, 1999, p. 278–292. 57

W. van der Hoek and M. de Rijke – "Generalized quantifiers and modal logic", *Journal of Logic, Language and Information* **2** (1993), no. 1, p. 19–58. 16, 93, 94

W. van der Hoek and M. de Rijke – "Counting objects", *Journal of Logic and Computation* **5** (1995), no. 3, p. 325–345. 16, 93, 94

J. van Eijck – "Constraint tableaux for hybrid logics", Tech. report, 2002. 57

A. van Gelder – "Problem generator `mkcnf`", 1993, Contributed to the DIMACS 1993 Challenge archive. 122

D. Westerståhl – "Quantifiers in formal and natural languages", Handbook of Philosophical Logic, Vol. IV (D. Gabbay and F. Guenthner, eds.), Dordrecht: Reidel, 1989, p. 1–1331. 93, 102